Elman Backpropagation as Reinforcement for Simple Recurrent Networks

André Grüning
gruening@sissa.it
Cognitive Neuroscience Sector, SISSA, 34014 Trieste, Italy

Simple recurrent networks (SRNs) in symbolic time-series prediction (e.g., language processing models) are frequently trained with gradient descent–based learning algorithms, notably with variants of backpropagation (BP). A major drawback for the cognitive plausibility of BP is that it is a supervised scheme in which a teacher has to provide a fully specified target answer. Yet agents in natural environments often receive summary feedback about the degree of success or failure only, a view adopted in reinforcement learning schemes.

In this work, we show that for SRNs in prediction tasks for which there is a probability interpretation of the network's output vector, Elman BP can be reimplemented as a reinforcement learning scheme for which the expected weight updates agree with the ones from traditional Elman BP. Network simulations on formal languages corroborate this result and show that the learning behaviors of Elman backpropagation and its reinforcement variant are very similar also in online learning tasks.

1 Introduction

Artificial neural networks arose as models of real neuronal networks. They are used as biologically inspired models of all kinds of brain processes, from the dynamics of networks of real neurons to the dynamics of cognitive processes, for which the model neurons correspond to assemblies of real neurons rather than to real neurons themselves (Ellis & Humphreys, 1999).

Generally network models are used in language modeling since they are thought of as more plausible than symbol-based approaches toward language and also because network models are often able to learn from scratch parts of language that were thought to be innate (Elman, 1990; Christiansen & Chater, 1999b). Thus, there is a natural interest in learning algorithms that are cognitively plausible. If they are also biologically plausible, this is even better, since then the learning algorithm fills the gap between what we know about cognitive processes and what we know about the underlying neural hardware of the brain (Jackendoff, 2002). van der Velde and de Kamps (2006), for example, proposed a neurobiologically based model of language processing, but hard-wired without learning. Thus, while the types of processing an artificial neural network can do are fairly well understood, it is more uncertain how networks can be trained on a particular task in a biologically and cognitively plausible way. The challenge is to find learning algorithms for artificial neural networks that are both efficient and plausible.

In cognitive science, the commonly used training algorithms are variants of gradient descent algorithms: for an input (or sequence of inputs), they produce an output that is compared to a target output. Some measure of error between actual output and expected output is then calculated. The gradient descent algorithm changes the weights in the network such that the error is minimized, typically following the negative gradient of the error as a function of all network parameters. This is a nontrivial task since at every synapse, we need information as to how much its weight contributes to the error (credit assignment problem). This is especially true when weight updates are to be computed from quantities that are locally available at the synapse the weight belongs to, that is, in a biologically plausible manner. Backpropagation and its variants bypass this problem by using each synapse bidirectionally: to forward-propagate activity and backpropagate each synapse's contribution to the total error.

Backpropagation (BP) and other standard gradient descent algorithms are thus implausible for at least two reasons: first, biologically, because synapses are used bidirectional; second, cognitively, because a full target with the correct output pattern has to be supplied to the network. In this work, we deal with the second implausibility: the need for a fully specified target answer.

In reinforcement learning (RL), the teacher instead of the fully specified target answer supplies only feedback about success or failure of an answer. This is cognitively more plausible than supervised learning since a fully specified correct answer might not always be available to the learner or even the teacher (Sutton & Barto, 2002; Wörgötter & Porr, 2005).

Standard reinforcement algorithms are less powerful than the gradient descent–based ones. This seems to be due to the lack of good solutions for the credit assignment problem for hidden-layer neurons (Roelfsema & van Ooyen, 2005). Thus, there is a trade-off: BP is quite efficient, but not so plausible cognitively, and vice versa for RL. Therefore, the objective of this letter is to find an algorithm that is both as efficient as BP and as plausible as RL. We want to concentrate on simple recurrent networks (SRNs) that are trained on a prediction task since these have been used extensively as models in the cognitive sciences, in particular to model language processing (Christiansen & Chater, 1999a).

The algorithm we propose shares properties with RL, such as a reward signal instead of a fully specified target. However, regarding its speed of learning and the types of solutions it finds, we will provide evidence that it essentially behaves like Elman BP. Our algorithm will not provide an improvement on the technical side. Rather, it is a reimplementation of Elman

BP in terms of RL. What is gained is an improvement in cognitive plausibility. Since the suggested algorithm will turn out to behave virtually like Elman BP, this letter can also be seen as a post hoc justification of the use of Elman BP in so many articles in cognitive science.

We use ideas from the attention-gated reinforcement learning (AGREL) learning scheme (Roelfsema & van Ooyen, 2005). The AGREL learning scheme effects an amalgamation of BP and RL for feedforward (FF) networks in classification tasks. However we need to recast these ideas in order to apply them to SRNs in prediction tasks. We call the resulting reimplementation of Elman BP as a reinforcement scheme *recurrent BP-as-reinforcement learning (rBPRL).*

We start with an outline of Elman BP since our suggested modifications build on an understanding of BP. Then we introduce modifications in order to reimplement Elman BP as a reinforcement scheme and show that in the sense of expectations, Elman BP and its reinforcement version, rBPRL, effect the same weight changes (for a different formulation, see Grüning, 2005). After these theoretical sections, we present simulations that corroborate the claimed agreement in practice.

2 SRNs and Elman BP

An SRN in its simplest form is a three-layer feedforward (FF) network; in addition, the hidden layer is self-recurrent (Elman, 1990). It is a special case of a general recurrent neural network (RNN) and could thus be trained with full backpropagation through time (BPTT) and its variants (Williams & Peng, 1990). However, instead of regarding the hidden layer as self-recurrent, one introduces a so-called context layer into which the activities of the hidden neurons are stored in each time step and which acts as an additional input to the hidden layer in the next time step. Regarding the forward-propagation of activity through the SRN, these two views are equivalent.

In order to keep notation simple, we will deal only with networks that are strictly layered: there are connections only between subsequent layers and assume that there is only one hidden layer. Generalizations where neurons receive input from other downstream layers and with more hidden layers are, of course, possible. Furthermore, we refrain from explicitly introducing a bias term, since its effect can easily be achieved by a unit with constant activation 1.

2.1 Forward Pass. Let $y^{(0)}(t)$, $y^{(1)}(t)$, and $y^{(2)}(t)$ denote the activation vectors of the input, hidden, and output layer, respectively, at (discrete) time step t and $a^{(r)}(t)$, $r = 1, 2$ the vectors of the corresponding net activations, and finally let $f : a \mapsto \frac{1}{1+e^{-a}}$ be the standard sigmoid and $w^{r,r'}$ denote the weight matrices between layer r and r'. Then the forward pass of activity through a three-layer SRN viewed as a special case of a general RNN can be expressed as follows:

$$a_i^{(1)}(t) = \sum_j w_{ij}^{1,0} y_j^{(0)}(t) + \sum_j w_{ij}^{1,1} y_j^{(1)}(t-1), \tag{2.1}$$

$$y_i^{(1)}(t) = f\big(a_i^{(1)}(t)\big), \tag{2.2}$$

$$a_i^{(2)}(t) = \sum_j w_{ij}^{2,1} y_j^{(1)}(t), \quad y_i^{(2)}(t) = f\big(a_i^{(2)}(t)\big). \tag{2.3}$$

When viewed as an FF network with extra input from the context layer, we set $\tilde{y}^{(0)}(t) := \begin{pmatrix} y^{(0)}(t) \\ y^{(1)}(t-1) \end{pmatrix}$ and $\tilde{w}^{1,0} := (w^{1,0}\ w^{1,1})$. This replaces equation 2.1 with

$$a_i^{(1)}(t) = \sum_j \tilde{w}_{ij}^{1,0} \tilde{y}_j^{(0)}(t), \tag{2.4}$$

making the SRN formally equivalent to an FF network with the addition of recycling the hidden-layer activations as part of the next input. For notational simplicity, we occasionally drop the explicit dependence of the variables on t when no confusion can arise.

For each input $y^{(0)}$, the layers are updated consecutively. Finally the output is read off from $y^{(2)}$ and the error E against a target vector u computed:

$$E(t) = \frac{1}{2} \sum_i \big(u_i(t) - y_i^{(2)}(t)\big)^2. \tag{2.5}$$

2.2 Elman Backpropagation. The SRN is viewed as an FF network with an additional set of inputs from the context layer. Hence, standard BP in conjunction with copying the hidden layer into the context layer (with immutable weights between the hidden and context layer equal to 1 and linear activation function) can be used for training (Elman, 1990). This scheme is called *Elman BP*. In fact, any algorithm suitable for FF networks will automatically be available for SRNs. We restate the main formulas of Elman BP in the following since the reinforcement implementation will be based on them.

For the update of the weights, we need to know how much each single weight $w_{ij}^{r,r'}$ contributes to the overall error E. For given input $y^{(0)}$ and target u, E can be viewed as a function of all weights w. In order to keep track of each weight's contribution $\frac{\partial E}{\partial w_{ij}^{r,r'}}$ to the error we first calculate the contribution $\Delta_i^{(r)} := \frac{\partial E}{\partial a_i^{(r)}}$ of each neuron's net input to the error. The Δs can be recursively computed layer-wise starting from the output layer, and from them, the weight updates δw with learning rate ϵ as

$$\Delta_i^{(2)}(t) = f'\big(a_i^{(2)}(t)\big)\big(y_i^{(2)} - u_i(t)\big), \tag{2.6}$$

$$\Delta_j^{(1)}(t) = f'\big(a_j^{(1)}(t)\big)\sum_i \Delta_i^{(2)}(t) w_{ij}^{2,1}, \tag{2.7}$$

$$\delta w_{ij}^{2,1}(t) := -\epsilon \Delta_i^{(2)}(t) y_j^{(1)}(t), \quad \delta \tilde{w}_{ij}^{1,0}(t) := -\epsilon \Delta_i^{(1)}(t) \tilde{y}_j^{(0)}(t), \tag{2.8}$$

since $\Delta_i^{(2)} y_j^{(1)} = \frac{\partial E}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial w_{ij}^{2,1}} = \frac{\partial E}{\partial w_{ij}^{2,1}}$ and analogously for $\delta\tilde{w}$. So weight changes follow the antigradient of error. Error contributions are not propagated further back; thus, Elman BP is a variant of BPTT(n), with $n = 1$ (Williams & Zipser, 1989).

The weight change δw might be applied to the weights in each time step ($w(t+1) = w(t) + \delta w(t)$, online learning) or collected and summed and applied to the weights only after a learning epoch has finished (batch learning). Since weight changes will take place at the end of an epoch or, ideally, on a timescale slower than the network dynamics, we will assume the weights to be fixed for the rest of the theory sections.

3 Prediction Tasks for SRNs

The tasks we are interested in are prediction tasks—predicting the next symbol in a temporal sequence of symbols that follow a certain rule or stochastic dependency. These tasks often arise in the context of modeling language or action sequences in cognitive science, or generally in predicting a stochastic process used as an information source. We formalize these notions following Tiňo and Dorffner (2001).

3.1 Information Source. Let A denote a finite alphabet with n elements, which, for the sake of notational simplicity, we assume to be $A = 1, 2, \ldots, n$. A stationary stochastic process (Cover & Thomas, 1991) produces a sequence $s_1 s_2, \ldots, \forall i : s_i \in A$ one symbol a time, so that up to time t, it has produced the finite sequence $\sigma_t := s_1 s_2, \ldots, s_t$, and is given by a family of time-invariant probability measures P_n on n-blocks (length n sequences over A) with the consistency condition that $\sum_{s\in A} P_{n+1}(\omega s) = P_n(\omega)$ for all n-blocks ω. Conditional probabilities are as usual defined as

$$P(s|\omega) = \frac{P_{n+1}(\omega s)}{P_n(\omega)} \tag{3.1}$$

and express the probability that an n-block ω will be followed by symbol s.

An example of a stochastic information source is a finite automaton (FA) that selects one of the state transitions of its current state randomly and emits the corresponding edge symbol (Hopcroft & Ullmann, 1979). However, the source can also be a more powerful automaton or (quantized) dynamical system.

The SRN's task is to predict the next symbol s_{t+1} of sequence σ_t, more precisely to infer (an approximation of) the conditional probabilities $P(s|\sigma_t)$ during training.

3.2 Input-Driven Systems of Iterated Functions. From the forward pass of activity, it is obvious that given $y^{(0)}(t)$, equations 2.1 and 2.2 can be rewritten as

$$y^{(1)}(t-1) \mapsto y^{(1)}(t) = F(y^{(0)}(t), y^{(1)}(t-1)) := \hat{f}(w^{1,0}y^{0}(t) + w^{1,1}y^{(1)}(t-1)), \tag{3.2}$$

where $\hat{f}$ denotes component-wise application of the standard sigmoid f. Especially in the case where there is only a finite set of different input vectors $y_s^{(0)}$ that encode the elements s of the finite alphabet A, it is more convenient to regard the dependence on the input symbol s as a parameter rather than a variable of F:

$$F_s(\cdot) := F\left(y_s^{(0)}, \cdot\right). \tag{3.3}$$

The forward pass of activity through the network is thus the application of a system of iterated functions that map the space of hidden-layer activations into itself driven by the source from which the symbols s are drawn (Blair & Pollack, 1997; Moore, 1998; Tiňo, Čerňanský, & Beňušková, 2004).

We finally introduce the notion F_σ to mean the resulting mapping when a sequence $\sigma = s_1 s_2, \ldots, s_n$ of input symbols is applied consecutively: $F_\sigma = F_{s_n} \circ F_{s_{n-1}} \circ, \ldots, \circ F_{s_1}$.

3.3 Standard Learning Schemes. After these formal prerequisites, let us look first at how the established learning schemes derive their error signals in order to reproduce the conditional next symbol probabilities in the network's output layer.

For each symbol s, there is an input neuron whose activation $y_s^{(0)}$ is set to one when this symbol is input, and to zero otherwise, and there is a corresponding output neuron whose activation $y_s^{(2)}$ (possibly after normalization) is a measure for the predicted probability of that symbol as a continuation of the input sequence. (Remember the symbols s are just natural numbers.) Finally let $y_0 := y^{(1)}(0)$ denote the (fixed) vector that the hidden-layer activities of the networks are initialized to before any symbols are entered.

3.3.1 Distribution Learning. A straightforward idea is to take as a target vector $u(t)$ directly the probability distribution $P(s|\sigma_t)$, that is,

$u_i(t) := P(i|\sigma_t)$ when the network has seen the initial sequence σ_t and thus is in state $y^{(1)}(t) = F_{\sigma_t}(y_0)$ with output activations $y^{(2)}(t)$ given by equation 2.3. In this case, equation 2.6 reads

$$\Delta_i^{(2)}(t) = f'\big(a_i^{(2)}(t)\big)\big(y_i^{(2)}(t) - P(i|\sigma_t)\big). \tag{3.4}$$

We refer to this scheme as distribution learning (DL). This type of learning might not be considered cognitively plausible, since it requires the teacher to know the family of probability distributions that define the information source.

3.3.2 Elman Learning. However, the conditional probabilities $P(s|\omega)$ might not be explicitly known to the teacher. So in this approach, after input sequence σ_t, the teacher waits until the source has produced s_{t+1}, and before using it as the next input symbol, it is presented as the new target vector; that is, $u(t)$ is a unit vector with the entry corresponding to s_{t+1} set to one and all others zero. Accordingly at time t, the expected target, averaging over all possible targets after sequence σ_t, is $E_{P(\cdot|\sigma_t)}(u_i(t)) = P(i|\sigma_t)$, and the expected value of the error signal according to equation 2.6 reads:

$$E_{P(\cdot|\sigma_t)}\big(\Delta_s^{(2)}(t)\big) = f'\big(a_i^{(2)}(t)\big)\big(y_i^{(2)}(t) - P(i|\sigma_t)\big), \tag{3.5}$$

so that the right-hand side agrees with equation 3.4, and the same is true for the other Δs recursively computed from this due to the linearity of equation 2.7 in the Δs, and finally also for the δws due to the linearity of equation 2.8.

This scheme is widely used in simulations in the cognitive and neurosciences. Its interesting feature is that the output activations will be driven toward the probability distribution $P(\cdot|\sigma_t)$ despite not explicitly being the target.

4 Elman BP as a Reinforcement Learning Scheme

Let us now recast Elman BP as an RL scheme in the following way. In prediction learning, the activation $y_i^{(2)}(t)$ of output i corresponds to the estimated probability of symbol i. Assume that the network is only allowed to select a single symbol k as a response in each time step. It selects this answer according to the distribution Q given by $y_i^{(2)}(t)/|y^{(2)}(t)|$, $|\cdot|$ denoting the one-norm. With network weights $w^{r,r'}$ and the initial state y_0 fixed, this distribution depends only on the input sequence σ_t the network has been driven with, so we write $Q(i|\sigma_t) := y_i^{(2)}(t)/|y^{(2)}(t)|$. The neuron k thus selected according to Q by some competitive process (Usher & McClelland, 2001) is called the winning neuron.

This answer k is compared to the target s_{t+1}, the actual next symbol drawn from the source according to $P(s|\sigma_t)$, and the network receives a reward $r(t) = 1$ only if $k = s_{t+1}$, and $r(t) = 0$ otherwise. The reward objectively to be expected after sequence σ_t and with winning neuron k is $E_{P(\cdot|\sigma_t)}(r(t)) = P(k|\sigma_t)$. The network compares the received reward $r(t)$ to the subjectively expected reward, namely, the activation $y_k^{(2)}(t)$ of the winning neuron (Tremblay & Schultz, 1999); the relative difference,

$$\delta(t) := \frac{y_k^{(2)}(t) - r(t)}{Q(k|\sigma_t)}, \tag{4.1}$$

serves as the direct reward signal for the network. From δ, we compute the error signals $\Delta^{(2)}$ for the output layer. Since attention is concentrated on the winning output k, only $\Delta_k^{(2)}(t)$ is different from zero, and we set

$$\Delta_i^{(2)}(t) := \begin{cases} 0, & i \neq k \\ f'\big(a_i^{(2)}(t)\big)\delta(t) & i = k, \end{cases} \tag{4.2}$$

overwriting the original definitions of the Δs as derivatives $\frac{\delta E}{\delta a_i^{(2)}}$. $\Delta^{(1)}$ and the weight updates δw are recursively computed from $\Delta^{(2)}$ formally as in equations 2.7 and 2.8. Since the reward $r(t)$ depends on s_{t+1} drawn according to P, we need to take expected values over P, with k still as the winning unit:

$$E_{P(\cdot|\sigma_t)}\big(\Delta_i^{(2)}(t)\big) = \begin{cases} 0 & i \neq k \\ f'\big(a_i^{(2)}(t)\big)\dfrac{y_i^{(2)}(t) - P(i|\sigma_t)}{Q(i|\sigma_t)} & i = k, \end{cases} \tag{4.3}$$

since $E_{P(\cdot|\sigma_t)}(r(t)) = P(k|\sigma_t)$ and all other quantities do not depend on s_{t+1}. However, k is also selected randomly according to Q, so we need to take the expected value over Q too. Since given the past input sequence σ_t, the distributions $P(\cdot|\sigma_t)$ and $Q(\cdot|\sigma_t)$ are independent, $E_{(Q,P)}(\cdot) = E_Q(E_P(\cdot))$, and we get

$$\begin{aligned} E_{Q(\cdot|\sigma_t)}\big(E_{P(\cdot|\sigma_t)}\big(\Delta_i^{(2)}(t)\big)\big) &= Q(i|\sigma_t) f'\big(a_i^{(2)}(t)\big)\frac{y_i^{(2)}(t) - P(i|\sigma_t)}{Q(i|\sigma_t)} \\ &= f'\big(a_i^{(2)}(t)\big)\big(y_i^{(2)}(t) - P(i|\sigma_t)\big), \end{aligned} \tag{4.4}$$

and this agrees with equations 3.4 and 3.5. By linearity the expected values of all other Δs and of the δws (and $\delta \tilde{w}$s) in this scheme coincide as well with their counterparts in standard Elman BP.

Compared to Elman BP, an error for a certain output is calculated only when it is the winning unit; it is updated less frequently by a factor $Q(i|\sigma_t)$. But $\Delta_i^{(2)}$ is larger by $1/Q(i|\sigma_t)$ to compensate for this.

Let us state briefly how the ingredients in this scheme fit into the standard reinforcement terminology (Sutton & Barto, 2002). First, there is a primary reward signal r from which a secondary one δ is derived as a comparison to the expected reward for a particular network state and selected action k. An action of our network is simply the selection of an output symbol k, and the value of this action k is the activation of the corresponding output neuron $y^{(2)}_k$. Finally actions k are selected (fairly nongreedily) on the basis of the distribution Q given by $\frac{y^{(2)}}{|y^{(2)}|}$. A policy is given by the inner dynamics of the network that maps an input $y^{(0)}$ and network state $y^{(1)}$ onto a set of action values and ultimately on the selected symbol. Furthermore, the dynamics of the network is a model of the information source.

We can conclude that we have found an RL scheme in which the expected weight updates are the same as in distribution or Elman learning.

5 From Expectations to Averages

In the preceding sections, we showed that expected values of the Δs and weight updates δw in Elman BP and rBPRL agree with their counterparts in DL. For practical applications, we need to say how these expected values can be approximated by averages from sampling one or more sequences from the information source.

Keeping the weights fixed as we did in the preceding paragraphs suggests a batch approach toward learning. However, we are ultimately interested in online learning (and Elman BP has often been used in such tasks) since they are also cognitively more plausible as no external reset of information source or network is needed and learning proceeds in an incremental way.

5.1 Batch Learning. For DL, there are no specific problems. An initial SRN state y_0 is fixed. The SRN is trained epoch-wise so that at the beginning of each epoch, the network state is reset to y_0 and time count to $t = 1$ and a new sequence drawn from the information source. The length of the epoch should be longer than the typical timescale of the source (if any and if known). For each target, the weight changes δw of equation 2.8 are collected and are applied at the end of the current epoch.

For Elman BP, since the target vectors are selected randomly according to $P(s|\sigma_t)$, we need to sample over a sufficient number of targets for the same σ_t before we update the weights to ensure a good approximation to the expectation of equation 3.5. Thus, within an epoch there will be several "subepochs" at the beginning of which network and time are reset and a new sequence drawn. Weight changes are collected during several subepochs and applied only on completion of an epoch. Again, the length of a subepoch should be longer than the typical timescale of the source, and the number of subepochs within an epoch should be such that each σ_t

appears a sufficient number of times so that the sampling of their respective targets becomes reliable enough for averaging.

For rBPRL, the procedure will be just like for Elman BP. Since the error signals Δ have an even bigger variance than in Elman BP, due to averaging for both P and Q, we need to sample the Δs even longer than before for the targets to yield reliable average Δs; thus, epochs will contain more subepochs.

The difficult point is to decide on the length of subepochs and their number when one knows nothing about the timescale of the source so that the network can generalize from the finite sequences it sees in training to longer sequences.

5.2 Online Learning. Insisting on cognitive plausibility, resetting the network to a fixed reset state y_0 seems unnatural, as does the epoch structure. Learning ought to be incremental just as new data are coming in. Let us therefore discuss the problems we incur in the transition from batch to online learning.

Online learning means that weight updates δw are applied after each target presentation, and this means that the network dynamics change constantly, so that in equation 3.5, $a^{(2)}(t)$ and $y^{(2)}(t)$, and in equations 4.3 and 4.4, $Q(k|\sigma_t)$ will be different for each presentation of the same sequence σ_t in a later subepoch and depend on the complete learning history in an intricate way, known as the moving targets problem. However, simulational results have shown for Elman BP and similar gradient descent algorithms that networks will approximate the averaged target for a sequence σ_t well when the learning rate is small (Williams & Peng, 1990), that is, the timescales of weight changes and the timescale of the source (or epoch) are decoupled, so that $y^{(2)}$ and Q will differ only slightly for presentation of the same sequence σ_t in different epochs. A rigorous convergence result for BPTT under certain conditions is derived in Kuan, Hornik, and White (1994). A proof for rBPRL or Elman BP would presumably have to follow the same line of argumentation.

Weights are changed incrementally in every time step now, but we have retained the (sub-)epoch structure insofar as we still reset the networks to y_0 and time count to $t = 1$ at the beginning of each epoch.

Many information sources can be predicted well when one bases one's predictions on only a (fixed) finite number of recent symbols in a sequence, though, of course, cases exist where this is not the case at all. When L_n denotes the cut-off operator, that is, $L_n(\sigma_t) = s_{t-n+1}s_{t-n+2}, \dots, s_n$ for $t > n$ and trivial operation for $t \leq n$, this means that the probabilities $P(s|\sigma_t)$ can be well approximated by the probabilities $P(s|L_n(\sigma_t))$. For Markov processes of order n, this approximation is exact. For fully fledged finite state processes, this approximation is exact except for a fraction of input histories whose probability measure decreases exponentially with n due to the guaranteed existence of a synchronizing word (Grüning, 2004; Lind & Marcus, 1995). Second, networks cannot be expected to reproduce the source exactly. Instead they have to build a model of the source using structure in the sequences (Cleeremans, Servan-Schreiber, & McClelland, 1989; Bodén & Wiles, 2000; Rodriguez, 2001; Grüning, 2006; Crutchfield, 1993; Tiňo & Dorffner, 2001). While no definite exact results exist about which models for which type of information source SRNs can evolve under gradient descent training, it is known that SRNs, when initialized with small weights, have an architectural bias toward definite memory machines (Hammer & Tiňo, 2003), that is, a model class with decaying memory. They tend to base their next symbol predictions on only a finite past since the last few input symbols have the greatest impact on the state-space activations. In addition, for the standard learning schemes, theoretical and heuristic results point into the same direction for bigger weights in the sense that under gradient descent learning, it is very hard to teach long-term dependencies to the networks (Bengio, Simard, & Frasconi, 1994).

Hence, also for rBPRL, we are led to assume that deviations caused when we do not reset the network at the beginning of each (sub)epoch to a definite value y_0 are negligible after a few input symbols.

If the information source is stationary and ergodic so that all finite subsequences will appear with their typical frequency in almost all long enough sequences, we can thus abolish the (sub-)epoch structure completely and train the network on a single long sequence from the source. This has the advantage too that we do not need to make assumptions about the internal timescale of the source to set the (sub-)epoch lengths.

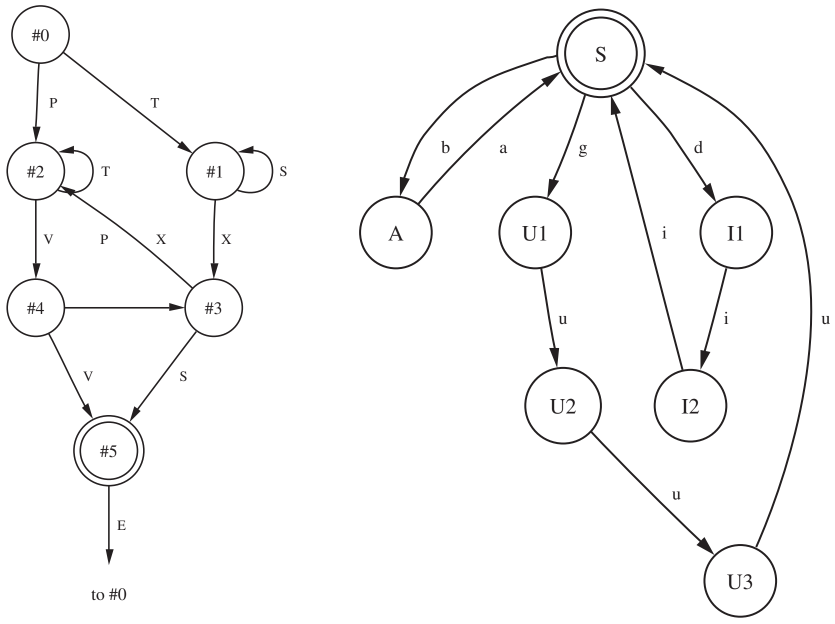
While there exists abundant simulational evidence that the network's output will approach the conditional probabilities of next symbols for DL and Elman BP also under online learning (Elman, 1990; Bodén & Wiles, 2000; Rodriguez, 2001; Grüning, 2006), this is of course no formal proof that rBPRL under these circumstances will converge to an error minimum too.

6 Simulations

In this section, we present some simulations corroborating the theoretical considerations. Our goal is to demonstrate that the modifications introduced to Elman BP in order to arrive at rBPRL have minor effect on the learning behavior, hence establishing also in practice that rBPRL is equivalent to Elman BP in the sense that it shows a similar learning speed and that the internal dynamics found as a solution to a certain task are similar too.

We have already shown that the expected error signals, and thus the weight updates, in rBPRL agree with the ones from Elman BP (and from DL). Thus, the speed of learning ought to be the same when all other parameters are kept constant and similar dynamics emerge.

While the learning behavior agrees in theory, in practice there are at least two factors that potentially introduce deviations between rBPRL and



(a) Reber language (b) *badiiguuu* language

Figure 1: FAs for the languages used in simulations. Double circles indicate the start states. The states carry arbitrary labels. Edge labels correspond to the symbol emitted when the edge is traversed.

Elman BP. First, we change weights online and not batch-like. This might have different influences for rBPRL and Elman BP on the strict validity of equations 3.5 and 4.4, respectively. For Elman BP, it has been found to work well. Second, due to averaging over both P and Q, the variance of the error signals that Δ in rBPRL is greater than that in Elman BP; thus, rBPRL is even more likely than Elman BP to leave narrow local minima. This is an advantage when the local minimum is shallow; however, rBPRL is also more likely to lose good local minima, which impedes learning. We will demonstrate here that one can balance these effects with a low enough learning rate ϵ.

We do so using two languages that have been used frequently as a benchmark in cognitively motivated symbolic time-series prediction. The first language is the so-called Reber language (Cleeremans et al., 1989; Reber, 1967) that can be generated by the FA in Figure 1a. Its main characteristic is that the same symbol can effect transitions between otherwise unrelated states. The second language is the *badiiguuu* language (Elman, 1990; Jacobsson, 2006), which needs to keep track of the number of symbols *i* and *u* in order to predict the probabilities of the next symbols (see Figure 1b).

For both languages, a stream of sentences is generated as follows. We start in the accept state and move to a new state along one of the outgoing transitions whose symbol is emitted. When there is more than one transition, each branch is chosen with equal probability. Then we move on to the next state and so forth. We complete a sentence when we reach the accept state again.[1]

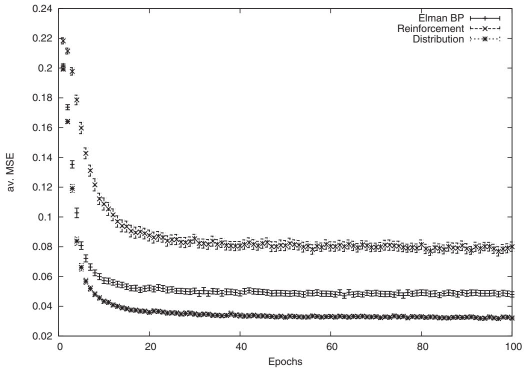
6.1 Learning Speed. Both the Reber language and *badiiguuu* language have six different symbols. Thus, for each, we create 50 SRNs with six input and output neurons and three neurons in the hidden layer. The weights in the 50 networks are initialized independently and identically distributed randomly from the interval $[-0.1, 0.1)$. We then make three identical copies of each of the 50 networks that are online trained with one of the following three training schemes, respectively: DL, standard Elman BP, and rBPRL.

The training consists of up to 500 epochs, where each epoch comprises a training phase of 100 sentences followed by a test phase of 50 sentences from the same online source; that is, all three copies of a network receive the same input in the same order starting from the same initial weights. Neither the network nor the information sources are reset at epoch boundaries; they are left running continuously, so epochs merely serve to organize the training. In the test phase, for each network the squared Euclidean distance between the actual output and the target distribution is calculated as a measure for error while weights are kept constant. Errors are then averaged over all patterns in a test epoch to yield the network's mean squared error (MSE) for that epoch.

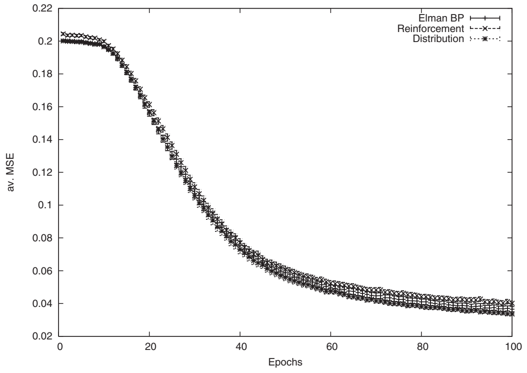
6.1.1 The Reber Language. Graphs for learning rates $\epsilon = 1.0, 0.1$ and 0.01 are displayed in Figure 2, showing the MSE averaged over the 50 independently randomly initialized network for each test epoch as a function of the number of training epochs.

For learning rate $\epsilon = 1.0$, we see that there is a wider distance between rBPRL and Elman BP, as well as a smaller one between Elman BP and DL, so that the error for DL finally is lowest while the error of rBPRL stays above Elman BP. This is true even for the initial epochs where larger error signals could facilitate finding a large but perhaps shallow local minimum. We conclude that the higher variance of error signals does play a role when learning rates ϵ are high, and thus leads to a dissociation of the error curves.

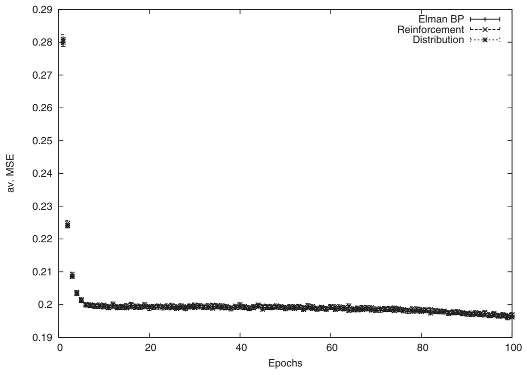
[1] Strictly speaking, these sources are not stationary and ergodic, since they start in a fixed state instead of drawing the start state each time from their stationary state distribution (Kitchens, 1998). However, each state distribution converges exponentially fast to the stationary distribution (in an irreducible, aperiodic FA) so that the differences are negligible after a few initial symbols.



(a) $\epsilon = 1.0$



(b) $\epsilon = 0.1$



(c) $\epsilon = 0.1$

Figure 2: Performance in terms of epoch-wise MSE of the Reber language averaged over the 50 networks for each language. Error bars indicate the standard error of the average MSE.

For $\epsilon = 0.1$ and $\epsilon = 0.01$, all three learning curves agree well. Thus, moving target problems and the higher variance of the error signal do not cause performance to deteriorate.

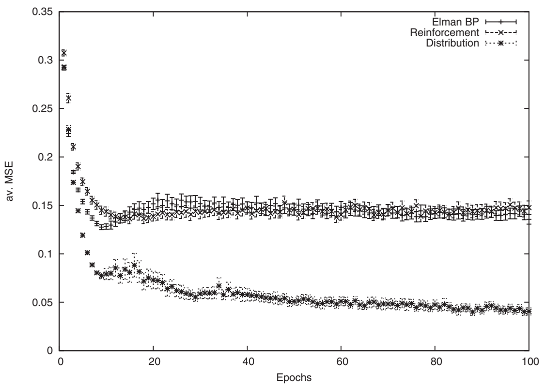
6.1.2 Badiiguuu. Averaged performance curves for *badiiguuu* are shown in Figure 3. While there is not much difference between the performance of Elman BP and rBPRL regardless of learning rate, there remains for all tested learning rates a certain gap between these two on the one side and DL on the other. However, we are mainly interested in the equivalence between Elman BP and rBPRL and can thus again state that their performance levels as functions of the number of training epochs are similar. We note that the error rate is generally higher than for the Reber languages. An inspection of the trained networks reveals that many of them fail to count the three *u* correctly.

6.2 Inner Dynamics. A main objective with this work is to give more plausibility to neural network simulations in cognitive modeling that make use of Elman BP for learning. The idea is to justify these simulations post hoc by demonstrating that Elman BP and rBPRL show similar learning behavior and thus that using Elman BP introduces only negligible deviations from rBPRL. Therefore, our main interest is in a comparison of Elman BP and rBPRL, and we do not pursue DL anymore in the following.

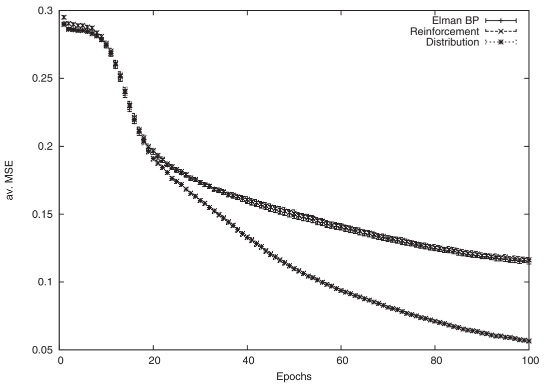
We have already presented some evidence that rBPRL and Elman BP learn equally fast (at least for low learning rates); however, "similar" means more than "equally fast." Therefore, we want to demonstrate that starting from identically initialized networks, the two learning algorithms also lead to the same internal dynamics or, put differently, that the solution space is identical for both algorithms for a given prediction task.

Thus, we also recorded the activations of the hidden layer during the test phases. For each test phase, we calculate the average squared Euclidean distance between the hidden-layer activations of the copies of the network trained with Elman BP and with rBPRL, respectively.

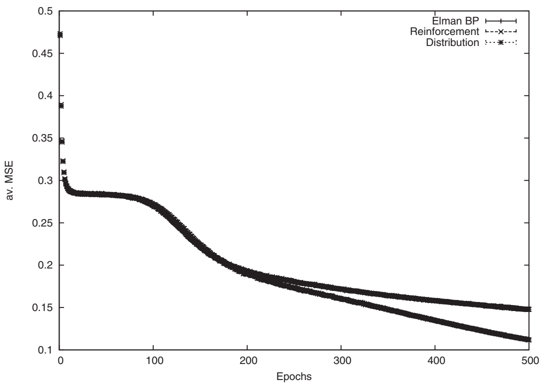
We start for both learning algorithms from identically initialized weight matrices, and hence from identical inner dynamics. When the learning rate is sufficiently low, the dynamics of copies of the network trained with different learning algorithms should also change slowly (bar any passing critical points in the dynamics, which usually is a rare event compared to continuous change of dynamics with parameters). When the dynamics stay sufficiently close to each other, the differences of hidden-layer activations between the two copies will also be small. When the dynamics evolve differently, the distance between the hidden-layer states will increase until the dynamics are not related any longer; that is, the activation distances reach a level that corresponds to the expected squared distance of two random vectors in the state space. For a three-dimensional state space $[0, 1]^3$, the expected squared distance of two random vectors drawn with uniform



(a) $\epsilon = 1.0$



(b) $\epsilon = 0.1$



(c) $\epsilon = 0.01$

Figure 3: Performance in terms of average MSE of the *badiiguuu* language over 50 networks. Error bars indicate the standard error as before.

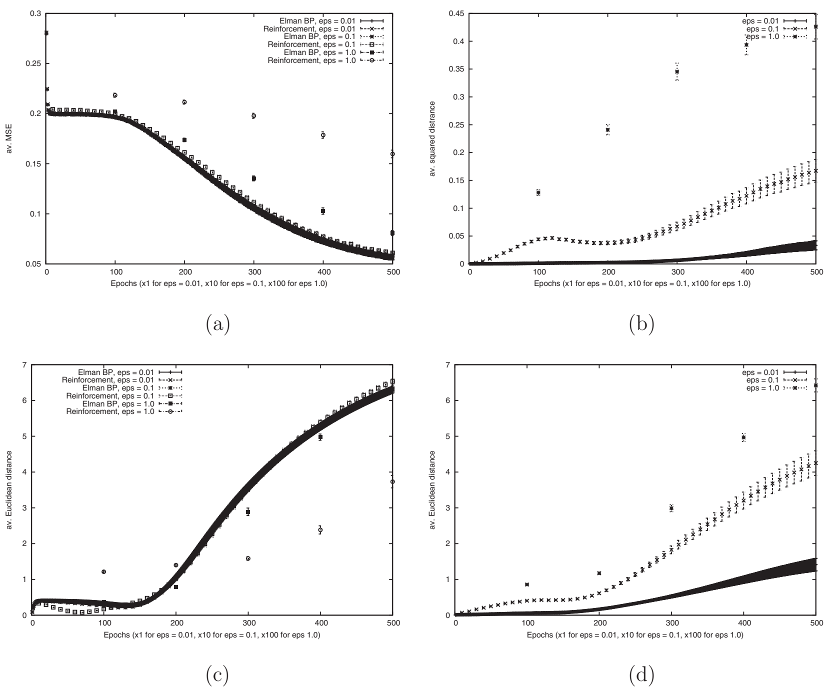


Figure 4: Reber language. In order to make the curves comparable for different learning rates and yield equal gross weight changes, the x-axes are tenfold stretched for $\epsilon = 0.1$ and hundred-fold for $\epsilon = 1.0$. Error bars indicate the standard error of the average mean squared error and average distances. (a) Comparison of performance curves. The networks for $\epsilon = 0.01$ and $\epsilon = 0.1$ perform equally after 10 times the number of epochs when trained with a 10 times smaller learning rate. (b) Differences of hidden-layer activations between Elman BP and rBPRL. At comparable performance and gross weight change, the differences are considerably smaller for smaller ϵ. (c) Euclidean distance of the weight matrix $w^{1,1}$ from the initial weight matrix as a function of epoch number. (d) Euclidean distance of the weight matrices $w^{1,1}$ between Elman BP and rBPRL.

probability is $\frac{1}{2}$. For higher learning rates, one expects this to be the case earlier than for lower learning rates, since in the latter case, the effect of the moving-targets problem as well as the effects of different variances of the weight updates, are lower.

When we compare the dynamics of networks across different learning rates we should also ensure that their performances are comparable. Figure 4a shows that the performances of networks trained on the Reber language with a learning rate $\epsilon = 0.01$ are comparable to those of networks trained with $\epsilon = 0.1$ for 10 times as long a training, and a bit less so to

$\epsilon = 1.0$ (for a hundred times as long a training), so that the total gross weight changes (number of epochs $\times$ learning rate) are of the same order of magnitude. Figure 4b shows the epoch-wise squared hidden-layer distances between Elman BP and rBPRL for learning rates $\epsilon = 1.0, 0.1, 0.01$, averaged over the 50 networks. A comparison across the different learning rates (aided by the epoch scaling) at comparable performance levels and comparable gross weight changes reveals that the dynamics for Elman BP and rBPRL become dissimilar much faster for $\epsilon = 1.0$ than for $\epsilon = 0.1$ or 0.01. In fact, the difference approaches its random value $\frac{1}{2}$ already after five epochs for $\epsilon = 1.0$, while it still stays considerably smaller for $\epsilon = 0.01$ than for $\epsilon = 0.1$.

A more direct way to compare the dynamics are the Euclidean distances of the weight matrices. In the following, we concentrate on the hidden-to-hidden matrix $w^{1,1}$ as a main ingredient of the state dynamics; the other weight matrices not presented here show a similar qualitative behavior. Figure 4c shows the Euclidean distance of $w^{1,1}$ to its value after initialization as a function of epochs. With scaling the epoch axis for $\epsilon = 0.1$ and $\epsilon = 1.0$, the average distances traveled from the initial matrix to the current one are comparable across learning algorithms and learning rates, with rBPRL for $\epsilon = 1.0$ deviating quite a bit. These distances show as a side effect that the matrices and thus dynamics have moved considerably away from the initial dynamics.[2] Finally Figure 4d compares epoch-wise the Euclidean distances of weight matrices $w^{1,1}$ between Elman BP and rBPRL across the different learning rates. The graphs show that the matrices become dissimilar faster for higher learning rates, comparable gross weight changes, and comparable distances from the initial matrix. For $\epsilon = 1.0$ the distances between the algorithms are of the same order of magnitude as the distance to the initial matrix, around 6 after five epochs, while for learning rates $\epsilon = 0.1$, the ratio of the distance to the initial matrix and the distance between the algorithms is roughly 1.5 after 50 epochs and 4 for $\epsilon = 0.01$, so that the matrix distances show that for low learning rates, the dynamics for Elman BP and rBPRL stay more similar for comparable distances from the initial weight matrix.

Figures 5a to 5d are the equivalent for the *badiiguuu* language and provide evidence for essentially the same behavior as for the Reber language. Also here, matrix distances from the initial matrix and between the algorithm variants show ratios about 1, 2, and 6 for $\epsilon = 1.0, 0.1, 0.01$. However, matrix distances are generally smaller, indicating that the dynamics have not moved as far from the initial matrix. This might also explain the generally lower performance of *badiiguuu* as compared to the Reber language.

[2] A similar picture emerges when comparing the maximal eigenvalue of $w^{1,1}$ across learning rates and algorithm variants. It rises from close to 0.15 to levels of about 6 after 500, 50, 5 epochs for $\epsilon = 0.01, 0.1, 1.0$, respectively, indicating that the contractive regime of the initial matrix has probably been left.

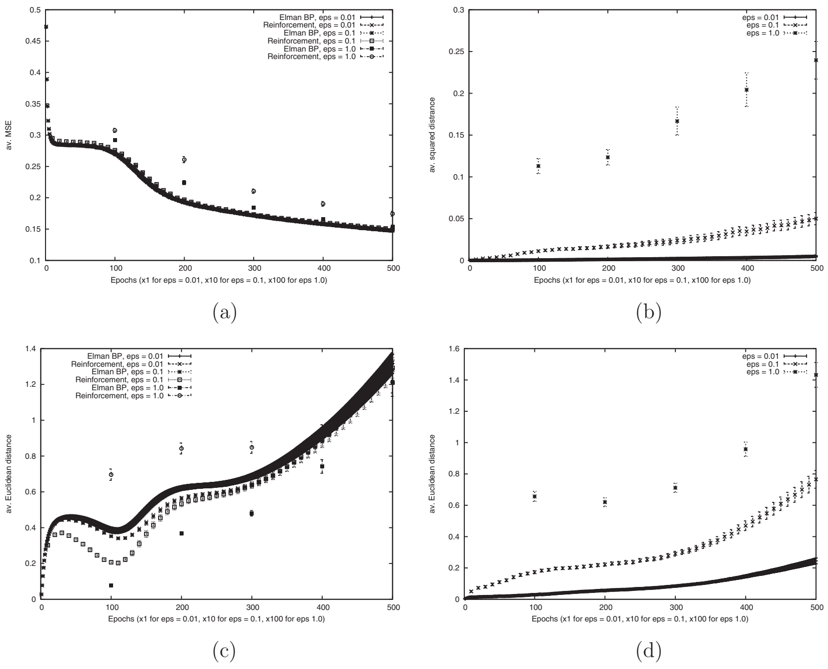


Figure 5: *badiiguuu* language. In order to make the curves comparable for different learning rates and yield equal gross weight changes, the x-axes are tenfold stretched for $\epsilon = 0.1$ and hundred-fold for $\epsilon = 0.01$. Error bars indicate the standard error. (a) Comparison of performance curves. The networks for $\epsilon = 0.01$ and $\epsilon = 0.1$ perform equally after 10 times the number of epochs when trained with a 10 times smaller learning rate. (b) Differences of hidden-layer activations between Elman BP and rBPRL. At comparable performance and gross weight change, the differences are considerably smaller for $\epsilon = 0.01$ than for 0.1 or 1.0. (c) Euclidean distance of the weight matrix $w^{1,1}$ from the initial weight matrix as a function of epoch number. (d) Euclidean distance of the weight matrices $w^{1,1}$ between Elman BP and rBPRL.

From both Figures 4 and 5, we can conclude that for too high a learning rate ϵ, the distances between the hidden-layer activations increase faster than for lower learning rates at comparable performance levels. Thus, the dynamics of two networks stay similar although trained with two different learning algorithms. This is an indication that, again for low learning rates ϵ, one can keep the search space for solutions the same for rBPRL and Elman BP. Even more so, given the same network initialization and identical training data, the two algorithms will lead to identical (or nearly identical) dynamical solutions for the learning task.

7 Discussion

The crucial facts why we can reimplement Elman BP as a reinforcement scheme are (1) a probability interpretation of the output vector, which allows us to regard the network as directing its attention to a single output that is stochastically selected and subsequently to relate the evaluative feedback to this single output with $\Delta \neq 0$ only for this output; and (2) the error contribution of each single neuron in the hidden layer to the total error is a linear superposition of the individual contributions of the neurons in the output layer in Elman BP (see equations 2.7 and 2.8). This enables us to concentrate on the contribution from a single output in each time step and still arrive at an expected weight update equal to the original Elman BP scheme.

Obviously the ideas laid out in this letter are applicable to all kinds of SRNs, multilayered or not, and more general RNNs as long as their recurrence can be treated in the sense of introducing context units with immutable copy weights, and they ought also to be applicable to other networks and gradient-based learning algorithms that fulfill the two above conditions (e.g., the long short-term memory algorithm; Hochreiter & Schmidhuber, 1997).

Our goal here is to make Elman BP cognitively and biologically more acceptable, not to offer a technical improvement of BP. Therefore a discussion of aspects of rBPRL's plausibility is in order.

First, we have replaced a fully specified target with a single evaluative feedback for SRNs in a symbolic prediction task, so that a more natural semisupervised RL scheme can be incorporated in Elman BP, yielding our rBPRL.

Second, look at the neural computability of the quantities we use in rBPRL. First, there is the relative difference δ of actual and expected reward. It can be realized as a prediction error neuron (Schultz, 1998; Wörgötter & Porr, 2005). Furthermore, attentive concentration on the winning neuron is physiologically plausible (Roelfsema & van Ooyen, 2005), and we are justified to inject a $\Delta \neq 0$ only for the winning neuron. Also the computation of Q (that also enters δ) from the output activations $y^{(2)}$ is not trivial. Technically it is just an addition of the $y_i^{(2)}$ and a division of each output activity by this sum. Positive evidence for pools of neurons doing precisely such a divisive normalization is summarized in Carandini and Heeger (1994). Furthermore, there needs to be a competitive process among the output neurons to finally select one according to Q_y as the winning neuron. Plausible brain mechanisms for this are discussed in Nowlan and Sejnowski (1995) and Usher and McClelland (2001). Finally, the Δs have to be calculated. The term that seems not straightforward to calculate is $f'(a)$. But here the choice of the standard sigmoid f as the activation function leads to a derivative that can be calculated from the function value $y = f(a)$ alone as $f'(a) = f(a)(1 - f(a)) = y(1 - y)$, so that this derivative is also natural to compute.

A third set of problems to be addressed is related to locality and how past values can be stored. In order to make all quantities for weight update available locally at a synapse, the weights w are used bidirectionally. And that is not really biologically plausible. When model neurons stand for assemblies of real neurons, the assumption of reciprocal connections of roughly equal strength can, however, be justified (Fellemann & van Essen, 1991). Therefore, Roelfsema and van Ooyen (2005) suggest a second set of weights $\bar{w}$ that are used in backward propagation of error, while w is used in the forward pass of activity only. This second set of weights $\bar{w}$ is initialized independently from w but updated, mutatis mutandis, according to equations 2.6 to 2.8, however, with a small noise term η with mean 0 added in order to introduce deviations between forward and backward connections:

$$\delta \bar{w}_{ij}^{r,r'} = -(1+\eta)\epsilon \Delta_i^{(r)} y_j^{(r')}. \tag{7.1}$$

In simulations with rBPRL and separate backpropagation weights $\bar{w}$, different initialization and introduction of gaussian noise (mean zero, standard deviation 0.2) as above did not change the average performance curves. Thus, rBPRL can cope with backpropagation weights that are not exactly equal to the forward-propagation weights.

In our view, the bidirectional use of weights or the introduction of a second set of weights in this explicit manner remains the weakest point of both Roelfsema and van Ooyen's AGREL and our rBPRL and needs further elaboration. An interesting route to pursue would be to incorporate elements of weight perturbation algorithms (Rowland, Maida, & Berkeley, 2006).

For SRNs we also need to make plausible that the activations of the hidden layer can be retrieved after one time step when they are to act as inputs to the hidden layer again. This assumption is again plausible in view of the ample recurrent connections in the brain, which might recycle activation from a neighboring cell for some time so that its activation can be traced.

Also in this case, we cannot expect that the activation can be retrieved with perfect accuracy. Hence, we conducted an additional set of simulations in which we added gaussian noise with mean zero and standard deviation 0.2 to the activations in the context layer. This modification left the performance curves for both the Reber and the *badiiguuu* languages unaltered, demonstrating that rBPRL can also cope with an imprecise context layer.

While our reinforcement scheme easily extends even to fully recurrent networks trained with BP through time (again, it is mainly a question of the linearity of the Δs), its naturalness would of course hinge on a mechanism that allows tracing a neuron's or assembly's activation with some precision for a longer time.

8 Conclusion

We have found an RL scheme that behaves essentially like the Elman BP scheme for SRNs in prediction tasks; however, it is cognitively more plausible by using a success-failure signal instead of a prescribed target. Essential in transforming Elman BP into a reinforcement scheme was (1) that the Elman BP error signal for the complete target is a linear superposition of the error for each single output neuron and (2) the probabilistic nature of the task: select one possible output randomly and direct the network's attention toward it until it is rewarded. We have shown that expected weight updates are the same for DL, Elman BP, and rBPRL under the assumption of fixed weights. Furthermore, we have found evidence in the simulations that DL, rBPRL, and Elman BP also behave similar in online learning with incremental weight updates (for low learning rates). They agree in terms of the number of epochs to reach a certain level of performance. Also, there is evidence that the solution spaces for rBPRL and Elman BP are essentially the same, at least for low learning rates ϵ.

Furthermore, we have briefly discussed the cognitive and biological plausibility of other ingredients in the recurrent BP-as-reinforcement scheme. It seems that there is good evidence for all of them at least in some parts of the brain, possibly with the exception of the second set of weights used for the backward propagation phase. Enhanced cognitive plausibility for Elman BP in the form of rBPRL thus gives SRN use in cognitive science a stronger standing.

Acknowledgments

I thank three anonymous reviewers for many profound and valuable comments and Thomas Wennekers, Arjen van Ooyen, and Alessandro Treves for stimulating discussions. This work was supported by Human Frontier grant RGP0047/2004-C.

References

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Blair, A. D., & Pollack, J. B. (1997). Analysis of dynamical recognizers. *Neural Computation*, *5*, 1127–1142.

Bodén, M., & Wiles, J. (2000). Context-free and context-sensitive dynamics in recurrent neural networks. *Connection Science*, *12*(3/4), 197–210.

Carandini, M., & Heeger, D. J. (1994). Summation and division by neurons in primate visual cortex. *Science*, *264*(5163), 1333–1336.

Christiansen, M. H., & Chater, N. (1999a). Connectionist natural language processing: The state of the art. *Cognitive Science*, *28*(4), 417–437.

Christiansen, M. H., & Chater, N. (1999b). Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, *23*(2), 157–205.

Cleeremans, A., Servan-Schreiber, D., & McClelland, J. L. (1989). Finite state automata and simple recurrent networks. *Neural Computation*, *1*, 372–381.

Cover, T. M., & Thomas, J. A. (1991). *Elements of information theory*. New York: Wiley.

Crutchfield, J. P. (1993). The calculi of emergence: Computation, dynamics, and induction. *Physika D*, *75*, 11–54.

Ellis, R., & Humphreys, G. (1999). *Connectionist psychology*. Hove, East Sussex: Psychology Press.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*, 179–211.

Fellemann, D., & van Essen, D. (1991). Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex*, *1*, 1–47.

Grüning, A. (2004). *Neural networks and the complexity of languages*. Unpublished doctoral dissertation, University of Leipzig.

Grüning, A. (2005). Back-propagation as reinforcement in prediction tasks. In W. Duch, J. Kacprzyk, E. Oja, & S. Zadrozny (Eds.), *Proceedings of the International Conference on Artificial Neural Networks* (pp. 547–552). Berlin: Springer.

Grüning, A. (2006). Stack- and queue-like dynamics in recurrent neural networks. *Connection Science*, *18*(1), 23–42.

Hammer, B., & Tiňo, P. (2003). Recurrent neural networks with small weights implement definite memory machines. *Neural Computation*, *15*(8), 1897–1929.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*, 1735–1780.

Hopcroft, J. E., & Ullmann, J. D. (1979). *Introduction to automata theory, languages, and computation*. Reading, MA: Addison-Wesley.

Jackendoff, R. (2002). *Foundations of language: Brain, meaning, grammar, evolution*. New York: Oxford University Press.

Jacobsson, H. (2006). The crystallizing substochastic sequential machine extractor—CrySSMEx. *Neural Computation*, *18*(9), 2211–2255.

Kitchens, B. P. (1998). *Symbolic dynamics*. Berlin: Springer.

Kuan, C.-M., Hornik, K., & White, H. (1994). A convergence result for learning in recurrent neural networks. *Neural Computation*, *6*, 420–440.

Lind, D., & Marcus, B. (1995). *An introduction to symbolic dynamics and coding*. Cambridge: Cambridge University Press.

Moore, C. (1998). Dynamical recognizers: Real-time language recognition by analog computers. *Theoretical Computer Science*, *201*, 99–136.

Nowlan, S. J., & Sejnowski, T. J. (1995). A selection model for motion processing in area MT of primates. *Journal of Neuroscience*, *15*(2), 1195–1214.

Reber, A. (1967). Implicit learning of artificial grammars. *Journal of Verbal Learning and Verbal Behavior*, *77*, 855–863.

Rodriguez, P. (2001). Simple recurrent networks learn context-free and context-sensitive languages by counting. *Neural Computation*, *13*, 2093–2118.

Roelfsema, P. R., & van Ooyen, A. (2005). Attention-gated reinforcement learning of internal representations for classification. *Neural Computation*, *17*, 1–39.

Rowland, B. A., Maida, A. S., & Berkeley, I. S. N. (2006). Synaptic noise as a means of implementing weight-perturbation learning. *Connection Science*, *18*(1), 69–79.

Schultz, W. (1998). Predictive reward signal of dopaminic neurons. *J. Neurophysiol.*, *80*, 1–27.

Sutton, R. S., & Barto, A. G. (2002). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Tiňo, P., Čerňanský, M., & Beňušková, Ľ. (2004). Markovian architectural bias of recurrent neural networks. *IEEE Transactions on Neural Networks*, *15*(1), 6–15.

Tiňo, P., & Dorffner, G. (2001). Predicting the future of discrete sequences from fractal representations of the past. *Machine Learning*, *45*(2), 187–217.

Tremblay, L., & Schultz, W. (1999). Relative reward preferences in primate orbitofrontal cortex. *Nature*, *398*, 704–708.

Usher, M., & McClelland, J. L. (2001). On the time course of perceptual choice: The leaky competing accumulator model. *Psychological Review*, *108*, 550–592.

van der Velde, F., & de Kamps, M. (2006). Neural blackboard architectures of combinatorial structures in cognition. *Behavioral and Brain Science*, *29*(1), 1–72.

Williams, R. J., & Peng, J. (1990). An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, *2*, 490–501.

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running full recurrent neural networks. *Neural Computation*, *1*, 270–280.

Wörgötter, F., & Porr, B. (2005). Temporal sequence learning, prediction, and control—a review of different models and their relation to biological mechanisms. *Neural Computation*, *17*(2), 245–319.

Received February 17, 2006; accepted January 30, 2007.