Memorable Work About Working Memory

Pietro Vertechi
Supervisor: Christian Machens

Abstract

How does the brain store information over short time scales? What are the neural computations underlying short-term memory? To tackle this issue, we will first give a brief description of the different models for neural networks, the basic "hardware" that the brain uses. Then, we will give a short account of the different approach that people have used to study short-term memory in neuroscience, both theoretical and experimental. Finally, starting from some recent work on a principled top down approach to design networks capable of tracking a variable efficiently, we will sketch an approach to designing and learning neural networks implementing working memory.

1 Neural networks

At a first, basic, cartoon-like description, the key computational hardware of the brain is a neural network: a group of cells, called neurons, connected by synapses. When a neuron receives a sufficient amount of external current, it produces a short and stereotyped depolarisation, called action potential, or spike. Such electrical signal will then travel through all the synapses starting from their neuron and will change (positively or negatively, depending on the type of synapse) the membrane voltage of the postsynaptic neurons. These basic neurophysiological facts led McCulloch and Pitts (see [14]) to propose a very simple threshold-linear dynamics of the neuronal state, namely:

$$x_i(t) = \operatorname{sgn}\left(\sum_j W_{ij} x_j(t) - b_i\right) \tag{1}$$

where x_j is the state of neuron j (1 if the neuron is active, 0 otherwise), whereas W_{ij} is the strength of the connection from neuron j to neuron i (it can be positive or negative) and b_i is some sort of threshold denoting the amount of excitation neuron i needs in order to spike. In the follow up, we will use a very similar model, where the quantity of interest is not this abstract "activity" of the neuron, but rather the firing rate of the neuron (that is to say, the frequency at which it emits spikes). Still there are two key differences:

- The firing rates are not a discrete quantity, they do not have to be 0 or 1 but can vary continuously.
- This rate model is a linear dynamical system and is therefore much easyer to treat analytically.

The neural network equation describing this model is:

$$\dot{r}_i(t) = \tau \left(\sum_j W_{ij} r_j(t) + I_i(t) - \mu r_i(t) \right) \tag{2}$$

meaning that the change in the firing rate of neuron i is proportional to the excitation he has received from some other neurons in the network, namely $\sum_j W_{ij} r_j(t)$, or from outside: $I_i(t)$, minus the term $\mu r_i(t)$, often referred to the leak, corresponding to the fact that, in the absence of external stimulus, neural activity tend to decay back to 0.

A key difference between the brain and standard computers is the ability to self-organise: synapses evolve through time, on a time-scale much slower than the network dynamics, thus changing both the structure and the function of the network. In 1949, the psychologist Donald Hebb proposed a model for synaptic adaptation in neural networks, often abbreviated as Cells that fire together, wire together. He suggested that the synaptic strength between two neurons adapts to the firing-rate correlation of its associated pre- and postsynaptic neurons (see [4]). This so-called Hebbian learning and its numerous adaptations are vastly influential in neural network research until today. However, what would be the computational purpose of such plasticity rule is far from clear: in what follows, we will explain the link between Hebbian-like plasticity rules and working memory.

2 Working memory

In our everyday life we continuously store (and retrieve) information for short time intervals: this ability is called working memory. When we speak, for example, we normally remember how a sentence started. Working memory can be studied in tasks in which a subject, generally an animal, needs to remember an item for a couple of seconds. In particular, in [17], Romo recorded neural activity in macaque prefrontal cortex in a delayed discrimination task (see figure 1).

Many brain circuits are known to maintain information over short periods of time in the firing of their neurons [13]. Such "persistent activity" is likely to arise through reverberation of activity due to recurrent synapses. While many recurrent network models have been designed that remain active after transient stimulation, such as hand-designed attractor networks [19, 12] or randomly generated reservoir networks [8, 11], how neural networks can *learn* to remain active is less well understood.

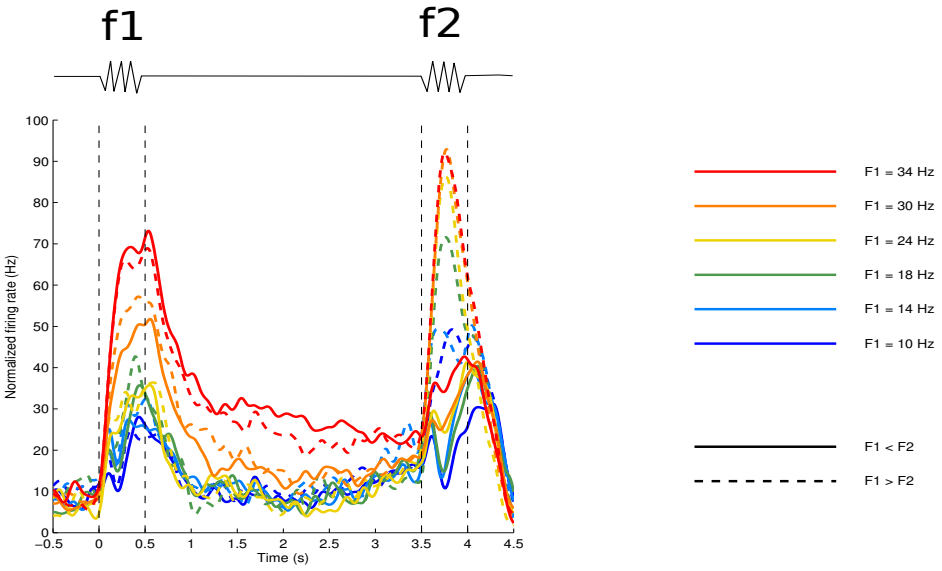

Figure 1: The monkey receives two stimulations at different times (indicated by the bumps in activity). This plot shows sustained firing rates in the delay period, monotonical in the first stimulation: the animal is remembering the value of the first input to compare it with the second, then forgets about it.

The problem of learning to remember the input history has mostly been addressed in supervised learning of recurrent networks. The classical approaches are based on backpropagation through time [20, 5]. However, apart from convergence issues, backpropagation through time is not a feasible method for biological systems. More recent work has drawn attention to random recurrent neural networks, which already provide a reservoir of time constants that allow to store and readout memories [8, 11]. Several studies have focused on the question of how to optimize such networks to the task at hand (see [10] for a review), however, the generality of the underlying learning rules is often not fully understood, since many rules are not based on analytical results or convergence proofs.

The unsupervised learning of short-term memory systems, on the other hand, is largely unchartered territory. While there have been several "bottom-up" studies that use biologically realistic learning rules and simulations (see e.g. [9]), we are not aware of any analytical results based on local learning rules.

Here we report substantial progress in following through a normative, "top-down" approach that results in a recurrent neural network with local synaptic plasticity. This network learns how to efficiently remember an input and its history. The learning rules are largely Hebbian or covariance-based, but separate recurrent and feedforward inputs. Our approach generalizes analogous work in the setting of efficient coding of an instantaneous signal, as developed in

[15, 18, 21, 3, 1].

3 The autoencoder

We start by recapitulating the autoencoder network shown in Fig. 2a. The autoencoder transforms a K-dimensional input signal, $\mathbf{x}$, into a set of N firing rates, $\mathbf{r}$, while obeying two constraints. First, the input signal should be reconstructable from the output firing rates. A common assumption is that the input can be recovered through a linear decoder, $\mathbf{D}$, so that

$$\mathbf{x} \approx \hat{\mathbf{x}} = \mathbf{D}\mathbf{r}. \tag{3}$$

Second, the output firing rates, $\mathbf{r}$, should provide an optimal or efficient representation of the input signals. This optimality can be measured by defining a cost $C(\mathbf{r})$ for the representation $\mathbf{r}$. For simplicity, we will in the following assume that the costs are quadratic (L2), although linear (L1) costs in the firing rates could easily be accounted for as well. We note that autoencoder networks are sometimes assumed to reduce the dimensionality of the input (undercomplete case, $N < K$) and sometimes assumed to increase the dimensionality (overcomplete case, $N > K$). Our results apply to both cases.

The optimal set of firing rates for a given input signal can then be found by minimizing the loss function,

$$L = \frac{1}{2} \left\| \mathbf{x} - \mathbf{D}\mathbf{r} \right\|^2 + \frac{\mu}{2} \left\| \mathbf{r} \right\|^2, \tag{4}$$

with respect to the firing rates $\mathbf{r}$. Here, the first term is the error between the reconstructed input signal, $\hat{\mathbf{x}} = \mathbf{D}\mathbf{r}$, and the actual stimulus, $\mathbf{x}$, while the second term corresponds to the "cost" of the signal representation. The minimization can be carried out via gradient descent, resulting in the differential equation

$$\dot{\mathbf{r}} = -\frac{\partial L}{\partial \mathbf{r}} = -\mu\mathbf{r} + \mathbf{D}^\top \mathbf{x} - \mathbf{D}^\top \mathbf{D}\mathbf{r}. \tag{5}$$

This differential equation can be interpreted as a neural network with a 'leak', $-\mu\mathbf{r}$, feedforward connections, $\mathbf{F} = \mathbf{D}^T$, and recurrent connections, $\mathbf{\Omega} = \mathbf{D}^\top\mathbf{D}$. The derivation of neural networks from quadratic loss functions was first introduced by Hopfield [6, 7], and the link to the autoencoder was pointed out in [18]. Here, we have chosen a quadratic cost term which results in a linear differential equation. Depending on the precise nature of the cost term, one can also obtain non-linear differential equations, such as the Cowan-Wilson equations [18, 7]. Here, we will first focus on linear networks, in which case 'firing rates' can be both positive and negative. Further below, we will also show how our results can be generalized to networks with positive firing rates and to networks in which neurons spike.

In the case of arbitrarily small costs, the network can be understood as implementing predictive coding [16]. The predicted input signal, $\mathbf{D}^\top\hat{\mathbf{x}} = \mathbf{D}^\top\mathbf{D}\mathbf{r}$,

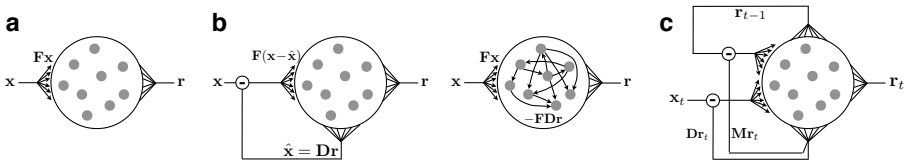

Figure 2: Autoencoders. (a) Feedforward network. The input signal $\mathbf{x}$ is multiplied with the feedforward weights $\mathbf{F}$. The network generates output firing rates $\mathbf{r}$. (b) Recurrent network. The left panel shows how the reconstructed input signal $\hat{\mathbf{x}} = \mathbf{Dr}$ is fed back and subtracted from the original input signal $\mathbf{x}$. The right panel shows that this subtraction can also be performed through recurrent connections $\mathbf{FD}$. For the optimal network, we set $\mathbf{F} = \mathbf{D}^\top$. (c) Recurrent network with delayed feedback. Here, the output firing rates are fed back with a delay. This delayed feedback acts as just another input signal, and is thereby re-used, thus generating short-term memory.

is subtracted from the actual input signal, $\mathbf{Dx}$, see Fig. 2b. Predictive coding here enforces a cancellation or 'balance' between the feedforward and recurrent synaptic inputs. If we assume that the actual input acts excitatory, for instance, then the predicted input is mediated through recurrent lateral inhibition. Recent work has shown that this cancellation can be mediated by the detailed balance of currents in spiking networks [2, 1], a result that deserves further investigation.

4 Unsupervised learning of the autoencoder with local learning rules

The transformation of the input signal, $\mathbf{x}$, into the output firing rate, $\mathbf{r}$, is largely governed by the decoder, $\mathbf{D}$, as can be seen in Eq. (5). When the inputs are drawn from a particular distribution, $p(\mathbf{x})$, such as the distribution of natural images or natural sounds, some decoders will lead to a smaller average loss and better performance. The average loss is given by

$$\langle L \rangle = \frac{1}{2} \left\langle \left\| \mathbf{x} - \mathbf{Dr} \right\|^2 + \mu \left\| \mathbf{r} \right\|^2 \right\rangle \tag{6}$$

where the angular brackets denote an average over many signal presentations. In practice, $\mathbf{x}$ will generally be centered and whitened. While it is straightforward to minimize this average loss with respect to the decoder, $\mathbf{D}$, biological networks face a different problem.[1] A general recurrent neural network is governed by the firing rate dynamics

$$\dot{\mathbf{r}} = -\mu \mathbf{r} + \mathbf{Fx} - \mathbf{\Omega r}, \tag{7}$$

and has therefore no access to the decoder, $\mathbf{D}$, but only to to its feedforward weights, $\mathbf{F}$, and its recurrent weights, $\mathbf{\Omega}$. Furthermore, any change in $\mathbf{F}$ and $\mathbf{\Omega}$

[1] Note that minimization of the average loss with respect to $\mathbf{D}$ requires either a hard or a soft normalization constraint on $\mathbf{D}$.

must solely relie on information that is locally available to each synapse.

We will assume that matrix $\mathbf{\Omega}$ is initially chosen such that the dynamical system is stable, in which case its equilibrium state is given by

$$\mathbf{F}\mathbf{x} = \mathbf{\Omega}\mathbf{r} + \mu\mathbf{r}. \tag{8}$$

If the dynamics of the input signal $\mathbf{x}$ are slow compared to the firing rate dynamics of the autoencoder, the network will generally operate close to equilibrium. We will assume that this is the case, and show that this assumption helps us to bridge from these firing rate networks to spiking networks further below.

A priori, it is not clear how to change the feedforward weights, $\mathbf{F}$, or the recurrent weights, $\mathbf{\Omega}$, since neither appears in the average loss function, Eq. (6). We might be inclined to solve Eq. (8) for $\mathbf{r}$ and plug the result into Eq. (6). However, we then have to operate on matrix inverses, the resulting gradients imply heavily non-local synaptic operations, and we would still need to somehow eliminate the decoder, $\mathbf{D}$, from the picture.

Here, we follow a different approach. We note that the optimal target network in the previous section implements a form of predictive coding. We therefore suggest a two-step approach to the learning problem. First, we fix the feedforward weights and we set up learning rules for the recurrent weights such that the network moves into a regime where the inputs, $\mathbf{F}\mathbf{x}$, are predicted or 'balanced' by the recurrent weights, $\mathbf{\Omega}\mathbf{r}$, see Fig. 2b. In this case, $\mathbf{\Omega} = \mathbf{F}\mathbf{D}$, and this will be our first target for learning. Second, once $\mathbf{\Omega}$ is learnt, we change the feedforward weights $\mathbf{F}$ to decrease the average loss even further. We then return to step 1 and iterate.

Since $\mathbf{F}$ is assumed constant in step 1, we can reach the target $\mathbf{\Omega} = \mathbf{F}\mathbf{D}$ by investigating how the decoder $\mathbf{D}$ needs to change. The respective learning equation for $\mathbf{D}$ can then be translated into a learning equation for $\mathbf{\Omega}$, which will directly link the learning of $\mathbf{\Omega}$ to the minimization of the loss function, Eq. (6). One thing to keep in mind, however, is that any change in $\mathbf{\Omega}$ will cause a compensatory change in $\mathbf{r}$ such that Eq. (8) remains fulfilled. These changes are related through the equation

$$\dot{\mathbf{\Omega}}\mathbf{r} + (\mathbf{\Omega} + \mu\mathbf{I})\dot{\mathbf{r}} = 0 \tag{9}$$

which is obtained by taking the derivative of Eq. (8) and remembering that $\mathbf{x}$ changes on much slower time scales, and can therefore be considered a constant. In consequence, we have to consider the combined change of the recurrent weights, $\mathbf{\Omega}$, and the equilibrium firing rate, $\mathbf{r}$, in order to reduce the average loss.

Let us assume a small change of $\mathbf{D}$ in the direction $\Delta\mathbf{D} = \epsilon\mathbf{x}\mathbf{r}^\top$, which is equivalent to simply decreasing $\mathbf{x}$ in the first term of Eq. (6). Such a small change can be translated into the following learning rule for $\mathbf{D}$,

$$\dot{\mathbf{D}} = \epsilon(\mathbf{x}\mathbf{r}^\top - \alpha\mathbf{D}), \tag{10}$$

where ϵ is sufficiently small to make the learning slow compared to the dynamics of the input signals $\mathbf{x} = \mathbf{x}(t)$. The 'weight decay' term, $-\alpha\mathbf{D}$, acts as a soft

normalization or regularizer on $\mathbf{D}$. In turn, to have the recurrent weights $\mathbf{\Omega}$ move towards $\mathbf{FD}$, we multiply with $\mathbf{F}$ from the left to obtain the learning rule[2]

$$\dot{\mathbf{\Omega}} = \epsilon(\mathbf{F}\mathbf{x}\mathbf{r}^\top - \alpha\mathbf{\Omega}). \tag{11}$$

Importantly, this learning rule is completely local: it only rests on information that is available to each synapse, namely the presynaptic firing rates, $\mathbf{r}$, and the postsynaptic input signal, $\mathbf{Fx}$.

Finally, we show that this learning rule decreases the loss function. As noted above, any change of $\mathbf{\Omega}$ causes a change in the equilibrium firing rate, see Eq. (9). By plugging the learning rule for $\mathbf{\Omega}$ into Eq. (9), and by remembering that $\mathbf{Fx} = \mathbf{\Omega}\mathbf{r} + \mu\mathbf{r}$, we obtain

$$\dot{\mathbf{r}} = -\epsilon\|\mathbf{r}\|^2\mathbf{r} + \epsilon\alpha(\mathbf{\Omega} + \mu\mathbf{I})^{-1}\mathbf{\Omega}\mathbf{r}. \tag{12}$$

Given that the second term on the right-hand-side is small against the first term, we conclude that, to first order, the firing rates decay in the direction of $\mathbf{r}$. In turn, the temporal derivative of the loss function,

$$\frac{d\langle L\rangle}{dt} = \left\langle (-\dot{\mathbf{D}}\mathbf{r} - \mathbf{D}\dot{\mathbf{r}})^\top(\mathbf{x} - \mathbf{D}\mathbf{r}) + \mu\dot{\mathbf{r}}^T\mathbf{r} \right\rangle \tag{13}$$

$$= \left\langle -\epsilon\|\mathbf{r}\|^2(\mathbf{x} - \mathbf{D}\mathbf{r})^\top(\mathbf{x} - \mathbf{D}\mathbf{r}) - \mu\epsilon\|\mathbf{r}\|^4 \right\rangle, \tag{14}$$

is always negative so that the learning rule for $\mathbf{\Omega}$ decreases the error. Note that the proof here rests on the parallelism of the learning of $\mathbf{D}$ and $\mathbf{\Omega}$. The decoder, $\mathbf{D}$, however, is merely a hypothetical quantity that does not have a physical counterpart in the network.

In step 2, we assume that the recurrent weights have reached their target, $\mathbf{\Omega} = \mathbf{FD}$, and we learn the feedforward weights. For that we notice that in the absolute minimum, as shown in the previous section, the feedforward weights become $\mathbf{F} = \mathbf{D}^\top$. Hence, the target for the feedforward weights should be the transpose of the decoder. Over long time intervals, the expected decoder is simply $\mathbf{D} = \langle \mathbf{x}\mathbf{r}^\top\rangle/\alpha$, since that is the fixed point of the decoder learning rule, Eq. (10). Hence, we suggest to learn the feedforward weights on a yet slower time scale $\beta \ll \epsilon$, according to

$$\dot{\mathbf{F}} = \beta(\mathbf{r}\mathbf{x}^\top - \lambda\mathbf{F}), \tag{15}$$

where $\lambda\mathbf{F}$ is once more a soft normalization factor. The fixed point of the learning rule is then $\mathbf{F} = \mathbf{D}^\top$. We emphasize that this learning rule is also local, based solely on the presynaptic input signal and postsynaptic firing rates.

In summary, we note that the autoencoder operates on four separate time scales. On a very fast, almost instaneous time scale, the firing rates run into equilibrium for a given input signal, Eq. (8). On a slower time scale, the input signal, $\mathbf{x}$, changes. On a yet slower time scale, the recurrent weights, $\mathbf{\Omega}$, are learnt, and their learning therefore uses many input signal values. On the final and slowest time scale, the feedforward weights, $\mathbf{F}$, are optimized.

[2]Note that the fixed point of the decoder learning rule is $\mathbf{D} = \langle \mathbf{x}\mathbf{r}^\top\rangle/\alpha$. Hence, the fixed point of the recurrent learning is $\mathbf{\Omega} = \mathbf{FD}$.

5 The autoencoder with memory

We are finally in a position to solve the problem we started out with, how to build a recurrent network that efficiently represents not just its present input, but also its past inputs. The objective function used so far, however, completely neglects the input history: even if the dimensionality of the input is much smaller than the number of neurons available to code it, the network will not try to use the extra 'space' available to remember the input history.

5.1 An objective function for short-term memory

Ideally, we would want to be able to read out both the present input and the past inputs, such that $\mathbf{x}_{t-n} \approx \mathbf{D}_n \mathbf{r}_t$, where n is an elementary time step, and $\mathbf{D}_n$ are appropriately chosen readouts. We will in the following assume that there is a matrix $\mathbf{M}$ such that $\mathbf{D}_n \mathbf{M} = \mathbf{D}_{n+1}$ for all n. In other words, the input history should be accessible via $\hat{\mathbf{x}}_{t-n} = \mathbf{D}_n \mathbf{r} = \mathbf{D}_0 \mathbf{M}^n \mathbf{r}_t$. Then the cost function we would like to minimize is a straightforward generalization of Eq. (4),

$$L = \frac{1}{2} \sum_{n=0} \gamma^n \|\mathbf{x}_{t-n} - \mathbf{D}\mathbf{M}^n \mathbf{r}_t\|^2 + \frac{\mu}{2} \|\mathbf{r}_t\|^2. \tag{16}$$

where we have set $\mathbf{D} = \mathbf{D}_0$.

Unfortunately, the direct minimization of this objective is impossible, since the network has no access to the past inputs $\mathbf{x}_{t-n}$ for $n \geq 1$. Rather, information about past inputs will have to be retrieved from the network activity itself. We can enforce that by replacing the past input signal at time t, with its estimate in the previous time step, which we will denote by a prime. In other words, instead of asking that $\mathbf{x}_{t-n} \approx \hat{\mathbf{x}}_{t-n}$, we ask that $\hat{\mathbf{x}}'_{(t-1)-(n-1)} \approx \hat{\mathbf{x}}_{t-n}$, so that the estimates of the input (and its history) are properly propagated through the network. Given the iterative character of the respective errors, $\|\hat{\mathbf{x}}'_{(t-1)-(n-1)} - \hat{\mathbf{x}}_{t-n}\| = \|\mathbf{D}\mathbf{M}^{n-1}(\mathbf{r}_{t-1} - \mathbf{M}\mathbf{r}_t)\|$, we can define a loss function for one time step only,

$$L = \frac{1}{2} \|\mathbf{x}_t - \mathbf{D}\mathbf{r}_t\|^2 + \frac{\gamma}{2} \|\mathbf{r}_{t-1} - \mathbf{M}\mathbf{r}_t\|^2 + \frac{\mu}{2} \|\mathbf{r}_t\|^2 . \tag{17}$$

Here, the first term enforces that the instantaneous input signal is properly encoded, while the second term ensures that the network is remembering past information. The last term is a cost term that makes the system more stable and efficient.

Note that a network which minimizes this loss function is maximizing its information content, even if the number of neurons, N, far exceeds the input dimension K, so that $N \gg K$. As becomes clear from inspecting the loss function, the network is trying to code an $N + K$ dimensional signal with only N neurons. Consequently, just as in the undercomplete autoencoder, all of its information capacity will be used.

5.2 Dynamics and learning

Conceptually, the loss function in Eq. (17) is identical to Eq. (4): we only need to vertically stack the feedforward input and the delayed recurrent input into a single high-dimensional vector $\mathbf{x}' = (\mathbf{x}_t \; ; \; \gamma \mathbf{r}_{t-1})$. Similarly, we can horizontally combine the decoder $\mathbf{D}$ and the 'time travel' matrix $\mathbf{M}$ into a single decoder matrix $\mathbf{D}' = (\mathbf{D} \;\; \gamma \mathbf{M})$. The above loss function then reduces to

$$L = \|\mathbf{x}'_t - \mathbf{D}'\mathbf{r}_t\|^2 + \mu \|\mathbf{r}_t\|^2, \tag{18}$$

and all of our derivations, including the learning rules, can be directy applied to this system. Note that the 'input' to the network now combines the actual input signal, $\mathbf{x}_t$, and the delayed recurrent input, $\mathbf{r}_{t-1}$. Consequently, this extended input is neither white nor centered, and we will need to work with the generalized dynamics and generalized learning rules derived in the previous section.

The network dynamics will initially follow the differential equation [3]

$$\mathbf{V} = \mathbf{F}\mathbf{x}_t + \mathbf{\Omega}^d \mathbf{r}_{t-1} - \mathbf{\Omega}^f \mathbf{r}_t - \mu \mathbf{r}_t \tag{19}$$

$$\dot{\mathbf{r}} = \mathbf{V} - \langle \mathbf{V} \rangle. \tag{20}$$

Compared to our previous network, we now have effectively three inputs into the network, the feedforward inputs with weight $\mathbf{F}$, a delayed recurrent input with weight $\mathbf{\Omega}^d$ and a fast recurrent input with weight $\mathbf{\Omega}^f$, see Fig. 2c. Consequently, there are also three learning rules: those for the fast recurrent weights, which follow the learning of $\mathbf{\Omega}$ above, and those for the feedforward and delayed recurrent weights, which follow the learning of $\mathbf{F}$ above. The optimal connectivities can be derived from the loss function and are

$$\mathbf{F} = \mathbf{D}^\top \tag{21}$$

$$\mathbf{\Omega}^d = \mathbf{M}^\top \tag{22}$$

$$\mathbf{\Omega}^f = \mathbf{D}^\top \mathbf{D} + \mathbf{M}^\top \mathbf{M}. \tag{23}$$

The learning rules follow from those in in the previous section and amount to

$$\dot{\mathbf{\Omega}}^f = \epsilon(\mathbf{F}\mathbf{x}_t \mathbf{r}_t^\top - \alpha \mathbf{\Omega}) \tag{24}$$

$$\dot{\mathbf{F}} = \beta\big((\mathbf{r}_t - \mathbf{F}\mathbf{x}_t)\mathbf{x}_t^\top - \alpha \mathbf{F}\big). \tag{25}$$

$$\dot{\mathbf{\Omega}}^d = \beta\big((\mathbf{r}_t - \mathbf{\Omega}^d \mathbf{r}_{t-1})\mathbf{r}_{t-1}^\top - \alpha \mathbf{\Omega}^d\big). \tag{26}$$

$$\tag{27}$$

6 Discussion

These results show a strong link between balancing and efficient coding: if neurons can compensate both external feedforward and delayed recurrent excitation

[3] We are now dealing with a delay-differential equation, which may be obscured by our notation. In practice, the term $\mathbf{r}_{t-1}$ would be replaced by a term of the form $\mathbf{r}(r - \tau)$, where τ is the actual value of the 'time step'.

with lateral inhibition, then, to some extent, they must be coding the temporal trajectory of the stimulus (to compensate for something, you must be coding it and if synapses are linear, so must be the decoder). This result can easily be learnt through synaptic plasticity of the lateral connections, using postsynaptic voltage or related quantities and is the most important part of the learning process. Performance can be further improved optimising feedforward connections (as well as the "time travel" matrix) to the input statistics, but that does not play as big a role, especially with gaussian uncorrelated input.

As much as this results may appear promising, many things are still left to do, in particular it would be very interesting to bring this whole formalism closer to biology. In this respect, two obvious improvements come to mind:

- Translate these results in more biophysically plausible networks, such as spiking networks of leaky integrate and fire neurons (it may be interesting to use the equivalence established in [1]).
- Generalise this framework to a setting in which not all the information has the same relevance: it would then be easier to check the model predictions with electrophysiological recordings in animal performing tasks, as they tend to remember particularly well those stimuli that they need to determine behaviour.

References

[1] D. G. Barrett, S. Denève, and C. K. Machens. Firing rate predictions in optimal balanced networks. In *Advances in Neural Information Processing Systems 26*, pages 1538–1546, 2013.

[2] M. Boerlin, C. K. Machens, and S. Denève. Predictive coding of dynamical variables in balanced spiking networks. *PLoS Computational Biology*, 9(11):e1003258, 2013.

[3] R. Bourdoukan, D. G. T. Barrett, C. K. Machens, and S. Denève. Learning optimal spike-based representations. In *Advances in Neural Information Processing Systems 25*, page epub. MIT Press, 2012.

[4] D. O. Hebb. *The Organisation of Behavior*. Psychology Press, 1949.

[5] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[6] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[7] J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Natl. Acad. Sci. USA*, 81:3088–3092, 1984.

[8] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. In *German National Research Center for Information Technology*, volume 48. 2001.

[9] A. Lazaar, G. Pipa, and J. Triesch. Sorn: a self-organizing recurrent neural network. *Frontiers in computational neuroscience*, 3:23, 2009.

[10] M. Lukoševičius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[11] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

[12] C. K. Machens, R. Romo, and C. D. Brody. Flexible control of mutual inhibition: A neural model of two-interval discrimination. *Science*, 307:1121–1124, 2005.

[13] G. Major and D. Tank. Persistent neural activity: prevalence and mechanisms. *Curr. Opin. Neurobiol.*, 14:675–684, 2004.

[14] W. S. Mcculloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[15] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, 37(23):3311–3325, 1997.

[16] R. P. N. Rao and D. H. Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2(1):79–87, 1999.

[17] R. Romo, C. D. Brody, A. Hernández, and L. Lemus. Neuronal correlates of parametric working memory in the prefrontal cortex. *Nature*, 399(6735):470–3, 1999.

[18] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen. Sparse coding via thresholding and local competition in neural circuits. *Neural computation*, 20(10):2526–2563, 2008.

[19] X.-J. Wang. Probabilistic decision making by slow reverberation in cortical circuits. *Neuron*, 36(5):955–968, 2002.

[20] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[21] J. Zylberberg, J. T. Murphy, and M. R. DeWeese. A sparse coding model with synaptically local plasticity and spiking neurons can account for the diverse shapes of v1 simple cell receptive fields. *PLoS Computational Biology*, 7(10):e1002250, 2011.