Due to additional references introduced in this appendix, we attach the entire reference list at the end.

Errata/Typos in the main text

First we'd like to point out some typos in the main text. None of them affects the core results presented in this work.

Major Typo. There is one important typo in the main text. One term is missing in Eq. 9/Left and Eq. 12/Left (there should be "$+\boldsymbol{w}^z \odot \hat{\boldsymbol{z}}(t)$" inside diag). We provide the correct/complete equations below (Eq. 36 and Eq. 21/Left, respectively). Please note that this is a pure printing mistake. The actual implementation is based on the correct equation, which is well tested by comparing the gradients computed through our RTRL algorithm against those computed by BPTT (using the common PyTorch reverse-mode automatic differentiation).

Minor Typos. There are also a few typos in the main text related to the range of indices that uses N (hidden dimension) where it should be D (input dimension). These are also not critical for the core description of our algorithms.

- **Line 140**, instead of "$\frac{\partial \boldsymbol{c}(t)}{\partial \boldsymbol{F}}, \frac{\partial \boldsymbol{c}(t)}{\partial \boldsymbol{Z}} \in \mathbb{R}^{N \times N \times N}$", the dimension should be $\mathbb{R}^{N \times N \times D}$
- **Line 143**, "For example for $\frac{\partial \boldsymbol{c}(t)}{\partial \boldsymbol{F}}$, we have, for $i, j, k \in \{1, ..., N\}$, " the range of j here is wrong, it should be "$j \in \{1, ..., D\}$"

We will fix them all in the final version.

A Derivations

A.1 RTRL for LSTM with Element-wise Recurrence

In Sec. 3.1, we only show RTRL equations (Eqs. 12) for one weight matrix of eLSTM ($\boldsymbol{F}$ in Eq. 5). Here we provide complete equations and their derivations for all other parameters.

First of all, the exact eLSTM architecture used in all our experiments use biases $\boldsymbol{b}^f, \boldsymbol{b}^z \in \mathbb{R}^N$, i.e., Eqs. 5 are replaced by:

$$\boldsymbol{f}(t) = \sigma(\boldsymbol{F}\boldsymbol{x}(t) + \boldsymbol{w}^f \odot \boldsymbol{c}(t-1) + \boldsymbol{b}^f) \quad ; \quad \boldsymbol{z}(t) = \tanh(\boldsymbol{Z}\boldsymbol{x}(t) + \boldsymbol{w}^z \odot \boldsymbol{c}(t-1) + \boldsymbol{b}^z) \tag{17}$$

Therefore, in addition to $\frac{\partial \boldsymbol{c}(t)}{\partial \boldsymbol{F}}, \frac{\partial \boldsymbol{c}(t)}{\partial \boldsymbol{Z}} \in \mathbb{R}^{N \times N \times D}$, and $\frac{\partial \boldsymbol{c}(t)}{\partial \boldsymbol{w}^f}, \frac{\partial \boldsymbol{c}(t)}{\partial \boldsymbol{w}^z} \in \mathbb{R}^{N \times N}$, we also have $\frac{\partial \boldsymbol{c}(t)}{\partial \boldsymbol{b}^f}, \frac{\partial \boldsymbol{c}(t)}{\partial \boldsymbol{b}^z} \in \mathbb{R}^{N \times N}$ as the sensitivity matrices to be computed/stored for RTRL. Note that adding these biases do not change the RTRL equations for $\boldsymbol{F}$ presented in Eq. 5/Sec. 3.1.

We recall that we define $\boldsymbol{e}(t) \in \mathbb{R}^N$ with $\boldsymbol{e}_i(t) = \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{c}_i(t)} \in \mathbb{R}$ for $i \in \{1, ..., N\}$ in Sec. 3.1, which can be computed using standard backpropagation (as we assume that we have no recurrent layer between $\boldsymbol{c}(t)$ and $\mathcal{L}(t)$).

For convenience, we also introduce three following intermediate variables $\hat{\boldsymbol{f}}(t), \hat{\boldsymbol{z}}(t), \hat{\boldsymbol{c}}(t) \in \mathbb{R}^N$ which appear in several equations.

$$\hat{\boldsymbol{f}}(t) = (\boldsymbol{c}(t-1) - \boldsymbol{z}(t)) \odot \boldsymbol{f}(t) \odot (1 - \boldsymbol{f}(t)) \tag{18}$$
$$\hat{\boldsymbol{z}}(t) = (1 - \boldsymbol{f}(t)) \odot (1 - \boldsymbol{z}(t)^2) \tag{19}$$
$$\hat{\boldsymbol{c}}(t) = \boldsymbol{f}(t) + \boldsymbol{w}^f \odot \hat{\boldsymbol{f}}(t) + \boldsymbol{w}^z \odot \hat{\boldsymbol{z}}(t) \tag{20}$$

The complete list of our RTRL equations is as follows.

Equations for $\boldsymbol{F}$. We define $\hat{\boldsymbol{F}}(t) \in \mathbb{R}^{N \times D}$ with $\hat{\boldsymbol{F}}_{i,j}(t) = \dfrac{\partial \boldsymbol{c}_i(t)}{\partial \boldsymbol{F}_{i,j}} \in \mathbb{R}$.

$$\hat{\boldsymbol{F}}(t) = \operatorname{diag}(\hat{\boldsymbol{c}}(t))\hat{\boldsymbol{F}}(t-1) + \hat{\boldsymbol{f}}(t) \otimes \boldsymbol{x}(t) \quad ; \quad \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{F}} = \operatorname{diag}(\boldsymbol{e}(t))\hat{\boldsymbol{F}}(t) \tag{21}$$

Equations for $\boldsymbol{Z}$. We define $\hat{\boldsymbol{Z}}(t) \in \mathbb{R}^{N \times D}$ with $\hat{\boldsymbol{Z}}_{i,j}(t) = \dfrac{\partial \boldsymbol{c}_i(t)}{\partial \boldsymbol{Z}_{i,j}} \in \mathbb{R}$.

$$\hat{\boldsymbol{Z}}(t) = \operatorname{diag}(\hat{\boldsymbol{c}}(t))\hat{\boldsymbol{Z}}(t-1) + \hat{\boldsymbol{z}}(t) \otimes \boldsymbol{x}(t) \quad ; \quad \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{Z}} = \operatorname{diag}(\boldsymbol{e}(t))\hat{\boldsymbol{Z}}(t) \tag{22}$$

Equations for $\boldsymbol{w}^f$. We define $\hat{\boldsymbol{w}}^f(t) \in \mathbb{R}^{N}$ with $\hat{\boldsymbol{w}}^f_i(t) = \dfrac{\partial \boldsymbol{c}_i(t)}{\partial \boldsymbol{w}^f_i} \in \mathbb{R}$.

$$\hat{\boldsymbol{w}}^f(t) = \hat{\boldsymbol{f}}(t) \odot \boldsymbol{c}(t-1) + \hat{\boldsymbol{c}}(t) \odot \hat{\boldsymbol{w}}^f(t-1) \quad ; \quad \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{w}^f} = \boldsymbol{e}(t) \odot \hat{\boldsymbol{w}}^f(t) \tag{23}$$

Equations for $\boldsymbol{w}^z$. We define $\hat{\boldsymbol{w}}^z(t) \in \mathbb{R}^{N}$ with $\hat{\boldsymbol{w}}^z_i(t) = \dfrac{\partial \boldsymbol{c}_i(t)}{\partial \boldsymbol{w}^z_i} \in \mathbb{R}$.

$$\hat{\boldsymbol{w}}^z(t) = \hat{\boldsymbol{z}}(t) \odot \boldsymbol{c}(t-1) + \hat{\boldsymbol{c}}(t) \odot \hat{\boldsymbol{w}}^z(t-1) \quad ; \quad \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{w}^z} = \boldsymbol{e}(t) \odot \hat{\boldsymbol{w}}^z(t) \tag{24}$$

Equations for $\boldsymbol{b}^f$. We define $\hat{\boldsymbol{b}}^f(t) \in \mathbb{R}^{N}$ with $\hat{\boldsymbol{b}}^f_i(t) = \dfrac{\partial \boldsymbol{c}_i(t)}{\partial \boldsymbol{b}^f_i} \in \mathbb{R}$.

$$\hat{\boldsymbol{b}}^f(t) = \hat{\boldsymbol{f}}(t) + \hat{\boldsymbol{c}}(t) \odot \hat{\boldsymbol{b}}^f(t-1) \quad ; \quad \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{b}^f} = \boldsymbol{e}(t) \odot \hat{\boldsymbol{b}}^f(t) \tag{25}$$

Equations for $\boldsymbol{b}^z$. We define $\hat{\boldsymbol{b}}^z(t) \in \mathbb{R}^{N}$ with $\hat{\boldsymbol{b}}^z_i(t) = \dfrac{\partial \boldsymbol{c}_i(t)}{\partial \boldsymbol{b}^z_i} \in \mathbb{R}$.

$$\hat{\boldsymbol{b}}^z(t) = \hat{\boldsymbol{z}}(t) + \hat{\boldsymbol{c}}(t) \odot \hat{\boldsymbol{b}}^z(t-1) \quad ; \quad \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{b}^z} = \boldsymbol{e}(t) \odot \hat{\boldsymbol{b}}^z(t) \tag{26}$$

Now we provide the corresponding derivations. Let $i, k \in \{1, ..., N\}$ and $j \in \{1, ..., D\}$.

Derivation for $\boldsymbol{F}$. Common to all cases, the goal is to compute the gradients of the loss w.r.t. the corresponding model parameter. Since we assume there is no recurrent layer after this layer (see our settings of Sec. 2), we can express the corresponding gradient as:

$$\frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{F}_{i,j}} = \sum_{k=1}^{N} \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{c}_k(t)} \times \frac{\partial \boldsymbol{c}_k(t)}{\partial \boldsymbol{F}_{i,j}} \tag{27}$$

As we assume we can compute the first factor $\dfrac{\partial \mathcal{L}(t)}{\partial \boldsymbol{c}_k(t)} = \boldsymbol{e}_k(t)$ using standard backpropagation, what remains to be computed is the second factor $\dfrac{\partial \boldsymbol{c}_k(t)}{\partial \boldsymbol{F}_{i,j}}$. The direct differentiation of Eq. 6 yields:

$$\frac{\partial \boldsymbol{c}_k(t)}{\partial \boldsymbol{F}_{i,j}} = (\boldsymbol{c}_k(t-1) - \boldsymbol{z}_k(t))\frac{\partial \boldsymbol{f}_k(t)}{\partial \boldsymbol{F}_{i,j}} + \boldsymbol{f}_k(t)\frac{\partial \boldsymbol{c}_k(t-1)}{\partial \boldsymbol{F}_{i,j}} + (1 - \boldsymbol{f}_k(t))\frac{\partial \boldsymbol{z}_k(t)}{\partial \boldsymbol{F}_{i,j}} \tag{28}$$

Now, we explicitly compute $\dfrac{\partial \boldsymbol{f}_k(t)}{\partial \boldsymbol{F}_{i,j}}$ and $\dfrac{\partial \boldsymbol{z}_k(t)}{\partial \boldsymbol{F}_{i,j}}$ as:

$$\frac{\partial \boldsymbol{f}_k(t)}{\partial \boldsymbol{F}_{i,j}} = \boldsymbol{f}'_k(t)\left(\boldsymbol{x}_j(t)\mathbb{1}_{k=i} + \boldsymbol{w}^f_k\frac{\partial \boldsymbol{c}_k(t-1)}{\partial \boldsymbol{F}_{i,j}}\right) ; \frac{\partial \boldsymbol{z}_k(t)}{\partial \boldsymbol{F}_{i,j}} = \boldsymbol{z}'_k(t)\boldsymbol{w}^z_k\frac{\partial \boldsymbol{c}_k(t-1)}{\partial \boldsymbol{F}_{i,j}} \tag{29}$$

where $\boldsymbol{f}'(t), \boldsymbol{z}'(t) \in \mathbb{R}^N$ are the "derivatives" of $\boldsymbol{f}(t)$ (sigmoid) and $\boldsymbol{z}(t)$ $(\tanh)$, i.e.,

$$\boldsymbol{f}'(t) = \boldsymbol{f}(t) \odot (1 - \boldsymbol{f}(t)) \quad ; \quad \boldsymbol{z}'(t) = 1 - \boldsymbol{z}(t)^2 \tag{30}$$

Now by substituting Eqs. 29 in Eq. 28, we obtain the following forward recursion equation:

$$\frac{\partial \boldsymbol{c}_k(t)}{\partial \boldsymbol{F}_{i,j}} = (\boldsymbol{c}_k(t-1) - \boldsymbol{z}_k(t))\boldsymbol{f}'_k(t)\boldsymbol{x}_j(t)\mathbb{1}_{k=i} \tag{31}$$

$$+ \left(\boldsymbol{f}_k(t) + (\boldsymbol{c}_k(t-1) - \boldsymbol{z}_k(t))\boldsymbol{f}'_k(t)\boldsymbol{w}^f_k + (1 - \boldsymbol{f}_k(t))\boldsymbol{z}'_k(t)\boldsymbol{w}^z_k\right) \frac{\partial \boldsymbol{c}_k(t-1)}{\partial \boldsymbol{F}_{i,j}} \tag{32}$$

Since the sensitivities are initialised by zero, i.e., $\dfrac{\partial \boldsymbol{c}_k(0)}{\partial \boldsymbol{F}_{i,j}} = 0$, and the additive term in Eq. 31 is non zero if and only if $k = i$, it follows that, for any t,

$$\frac{\partial \boldsymbol{c}_k(t)}{\partial \boldsymbol{F}_{i,j}} = 0 \quad \text{if} \quad k \neq i. \tag{33}$$

The other entries (the case where $k = i$) can be compactly represented using intermediate variables introduced above ($\hat{\boldsymbol{f}}(t)$, $\hat{\boldsymbol{z}}(t)$, $\hat{\boldsymbol{c}}(t)$; Eqs. 18-20), as follows:

$$\frac{\partial \boldsymbol{c}_i(t)}{\partial \boldsymbol{F}_{i,j}} = (\boldsymbol{c}_i(t-1) - \boldsymbol{z}_i(t))\boldsymbol{f}'_i(t)\boldsymbol{x}_j(t)$$

$$+ \left(\boldsymbol{f}_i(t) + (\boldsymbol{c}_i(t-1) - \boldsymbol{z}_i(t))\boldsymbol{f}'_i(t)\boldsymbol{w}^f_i + (1 - \boldsymbol{f}_i(t))\boldsymbol{z}'_i(t)\boldsymbol{w}^z_i\right) \frac{\partial \boldsymbol{c}_i(t-1)}{\partial \boldsymbol{F}_{i,j}} \tag{34}$$

$$= \hat{\boldsymbol{f}}_i(t)\boldsymbol{x}_j(t) + \left(\boldsymbol{f}_i(t) + \hat{\boldsymbol{f}}_i(t)\boldsymbol{w}^f_i + \hat{\boldsymbol{z}}_i(t)\boldsymbol{w}^z_i\right) \frac{\partial \boldsymbol{c}_i(t-1)}{\partial \boldsymbol{F}_{i,j}} \tag{35}$$

$$= \hat{\boldsymbol{f}}_i(t)\boldsymbol{x}_j(t) + \hat{\boldsymbol{c}}_i(t)\frac{\partial \boldsymbol{c}_i(t-1)}{\partial \boldsymbol{F}_{i,j}} \tag{36}$$

Finally, by introducing the notation $\hat{\boldsymbol{F}}(t) \in \mathbb{R}^{N \times D}$ with $\hat{\boldsymbol{F}}_{i,j}(t) = \dfrac{\partial \boldsymbol{c}_i(t)}{\partial \boldsymbol{F}_{i,j}} \in \mathbb{R}$, we arrive at Eq. 21/Left. Eq. 21/Right for the loss gradients is obtained by simplifying Eq. 27 through Eq. 33 (the sum reduces to one term).

The derivation is similar for $\boldsymbol{Z}$. We now show the derivation for $\boldsymbol{w}^f$.

Derivation for $\boldsymbol{w}^f$. Starting over, the goal is to compute the following gradient:

$$\frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{w}^f_i} = \sum_{k=1}^{N} \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{c}_k(t)} \times \frac{\partial \boldsymbol{c}_k(t)}{\partial \boldsymbol{w}^f_i} = \sum_{k=1}^{N} \boldsymbol{e}_k(t)\frac{\partial \boldsymbol{c}_k(t)}{\partial \boldsymbol{w}^f_i} \tag{37}$$

The direct differentiation through Eq. 6 yields:

$$\frac{\partial \boldsymbol{c}_k(t)}{\partial \boldsymbol{w}^f_i} = (\boldsymbol{c}_k(t-1) - \boldsymbol{z}_k(t))\frac{\partial \boldsymbol{f}_k(t)}{\partial \boldsymbol{w}^f_i} + \boldsymbol{f}_k(t)\frac{\partial \boldsymbol{c}_k(t-1)}{\partial \boldsymbol{w}^f_i} + (1 - \boldsymbol{f}_k(t))\frac{\partial \boldsymbol{z}_k(t)}{\partial \boldsymbol{w}^f_i} \tag{38}$$

Now, we explicitly compute $\dfrac{\partial \boldsymbol{f}_k(t)}{\partial \boldsymbol{w}^f_i}$ and $\dfrac{\partial \boldsymbol{z}_k(t)}{\partial \boldsymbol{w}^f_i}$, which yields:

$$\frac{\partial \boldsymbol{f}_k(t)}{\partial \boldsymbol{w}^f_i} = \boldsymbol{f}'_k(t)\left(\boldsymbol{c}_k(t-1)\mathbb{1}_{k=i} + \boldsymbol{w}^f_k\frac{\partial \boldsymbol{c}_k(t-1)}{\partial \boldsymbol{w}^f_i}\right) ; \ \frac{\partial \boldsymbol{z}_k(t)}{\partial \boldsymbol{w}^f_i} = \boldsymbol{z}'_k(t)\boldsymbol{w}^z_k\frac{\partial \boldsymbol{c}_k(t-1)}{\partial \boldsymbol{w}^f_i} \tag{39}$$

Now by substituting Eqs. 39 in Eq. 38, we obtain the following forward recursion equation:

$$\frac{\partial \boldsymbol{c}_k(t)}{\partial \boldsymbol{w}^f_i} = (\boldsymbol{c}_k(t-1) - \boldsymbol{z}_k(t))\boldsymbol{f}'_k(t)\boldsymbol{c}_k(t-1)\mathbb{1}_{k=i} \tag{40}$$

$$+ \left(\boldsymbol{f}_k(t) + (\boldsymbol{c}_k(t-1) - \boldsymbol{z}_k(t))\boldsymbol{f}'_k(t)\boldsymbol{w}^f_k + (1 - \boldsymbol{f}_k(t))\boldsymbol{z}'_k(t)\boldsymbol{w}^z_k\right) \frac{\partial \boldsymbol{c}_k(t-1)}{\partial \boldsymbol{w}^f_i} \tag{41}$$

Similar to the derivation for $\boldsymbol{F}$, as the sensitivities are initially zero, i.e., $\dfrac{\partial \boldsymbol{c}_k(t-1)}{\partial \boldsymbol{w}_i^f} = 0$, and non-zero terms are added only to the entries where $k = i$, it follows that for any t, $\dfrac{\partial \boldsymbol{c}_k(t)}{\partial \boldsymbol{w}_i^f} = 0$ if $k \neq i$, and the non-zero entries are:

$$\frac{\partial \boldsymbol{c}_i(t)}{\partial \boldsymbol{w}_i^f} = \hat{\boldsymbol{f}}_i(t)\boldsymbol{c}_i(t-1) + \hat{\boldsymbol{c}}_i(t)\frac{\partial \boldsymbol{c}_i(t-1)}{\partial \boldsymbol{w}_i^f} \tag{42}$$

which finally yields Eq. 23.

The derivation is almost the same for $\boldsymbol{b}^f$, except that, instead of $\boldsymbol{c}_k(t-1)\mathbb{1}_{k=i}$ in Eq. 39, we obtain $\mathbb{1}_{k=i}$. Finally, the derivations are also similar for $\boldsymbol{w}^z$ and $\boldsymbol{b}^z$.

A.2 Tractable RTRL for Certain Linear Transformers/Fast Weight Programmers

While the main focus of this work is to evaluate tractable RTRL using eLSTM, we also discuss that there are several neural architectures with tractable RTRL in the one-layer case. Here we show that RTRL is also tractable for a certain "simple" one-layer Linear Transformers/Fast Weight Programmers (FWPs; [15, 16, 17]).

The FWP in question transforms an input $\boldsymbol{x}(t) \in \mathbb{R}^N$ to an output $\boldsymbol{y}(t) \in \mathbb{R}^N$ at each time step t, as follows:

$$[\boldsymbol{k}(t), \boldsymbol{v}(t), \boldsymbol{q}(t)] = [\boldsymbol{K}\boldsymbol{x}(t), \boldsymbol{V}\boldsymbol{x}(t), \boldsymbol{Q}\boldsymbol{x}(t)] \tag{43}$$

$$\boldsymbol{W}(t) = \boldsymbol{W}(t-1) + \boldsymbol{v}(t) \otimes \boldsymbol{k}(t) \tag{44}$$

$$\boldsymbol{y}(t) = \boldsymbol{W}(t)\sigma(\boldsymbol{q}(t)) \tag{45}$$

where $\boldsymbol{k}(t) \in \mathbb{R}^N$, $\boldsymbol{v}(t) \in \mathbb{R}^N$, $\boldsymbol{q}(t) \in \mathbb{R}^N$, $\boldsymbol{W}(t) \in \mathbb{R}^{N \times N}$, with trainable parameters $\boldsymbol{K} \in \mathbb{R}^{N \times N}$, $\boldsymbol{V} \in \mathbb{R}^{N \times N}$, and $\boldsymbol{Q} \in \mathbb{R}^{N \times N}$ (we use the same dimension N everywhere for simplicity, but the following derivation remains valid for the general case where $\boldsymbol{x}(t)$ is of dimension D; the only requirement is that $\boldsymbol{k}(t)$ and $\boldsymbol{q}(t)$ are of the same dimension).

Let $i, j \in \{1, .., N\}$. Using the same definition of the loss $\mathcal{L}(t)$ defined in Sec. 2, the goal is to compute $\dfrac{\partial \mathcal{L}(t)}{\partial \boldsymbol{Q}_{i,j}}$, $\dfrac{\partial \mathcal{L}(t)}{\partial \boldsymbol{K}_{i,j}}$, $\dfrac{\partial \mathcal{L}(t)}{\partial \boldsymbol{V}_{i,j}}$. Since $\boldsymbol{q}(t)$ is not involved in the recurrent loop (here we are again under the one-layer assumption), the gradients $\dfrac{\partial \mathcal{L}(t)}{\partial \boldsymbol{Q}_{i,j}}$ can be computed using standard backpropagation. Hence, we focus on $\dfrac{\partial \mathcal{L}(t)}{\partial \boldsymbol{K}_{i,j}}$ and $\dfrac{\partial \mathcal{L}(t)}{\partial \boldsymbol{V}_{i,j}}$. Here we provide the RTRL derivation for $\dfrac{\partial \mathcal{L}(t)}{\partial \boldsymbol{K}_{i,j}}$ (the derivation for $\dfrac{\partial \mathcal{L}(t)}{\partial \boldsymbol{V}_{i,j}}$ is analogous). Let l, m, i and j denote positive integers. As in Sec. 2, it is practical to write down each element $\boldsymbol{W}_{l,m}(t) \in \mathbb{R}$ of $\boldsymbol{W}(t) \in \mathbb{R}^{N \times N}$, as well as each element $\boldsymbol{k}_m(t) \in \mathbb{R}$ of $\boldsymbol{k}(t) \in \mathbb{R}^N$, for all $l, m \in \{1, ..., N\}$:

$$\boldsymbol{W}_{l,m}(t) = \boldsymbol{W}_{l,m}(t-1) + \boldsymbol{v}_l(t)\boldsymbol{k}_m(t) \tag{46}$$

$$\boldsymbol{k}_m(t) = \sum_{n=1}^{N} \boldsymbol{K}_{m,n}\boldsymbol{x}_n(t) \tag{47}$$

The goal is to compute the following gradient:

$$\frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{K}_{i,j}} = \sum_{l=1}^{N}\sum_{m=1}^{N} \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{W}_{l,m}(t)} \times \frac{\partial \boldsymbol{W}_{l,m}(t)}{\partial \boldsymbol{K}_{i,j}} \tag{48}$$

The first factor $\dfrac{\partial \mathcal{L}(t)}{\partial \boldsymbol{W}_{l,m}(t)}$ can be computed using standard backpropagation (no recurrence involved). We derive a forward recursion formula for the second factor by differentiating Eq. 46:

$$\frac{\partial \boldsymbol{W}_{l,m}(t)}{\partial \boldsymbol{K}_{i,j}} = \frac{\partial \boldsymbol{W}_{l,m}(t-1)}{\partial \boldsymbol{K}_{i,j}} + \boldsymbol{v}_l(t)\frac{\partial \boldsymbol{k}_m(t)}{\partial \boldsymbol{K}_{i,j}} \tag{49}$$

Now, differentiating Eq. 47 yields:

$$\frac{\partial \boldsymbol{k}_m(t)}{\partial \boldsymbol{K}_{i,j}} = \begin{cases} 0 & \text{if} \quad m \neq i \\ \dfrac{\partial \boldsymbol{k}_i(t)}{\partial \boldsymbol{K}_{i,j}} = \boldsymbol{x}_j(t) & \text{Otherwise} \end{cases} \tag{50}$$

This last equation is a source of complexity reduction: in the 3-dimentional tensor $\frac{\partial \boldsymbol{k}_m(t)}{\partial \boldsymbol{K}_{i,j}}$, many terms are zero, and non-zero component only depends on j. Eq. 49 becomes:

$$\frac{\partial \boldsymbol{W}_{l,m}(t)}{\partial \boldsymbol{K}_{i,j}} = \begin{cases} 0 & \text{if} \quad m \neq i \\ \dfrac{\partial \boldsymbol{W}_{l,i}(t-1)}{\partial \boldsymbol{K}_{i,j}} + \boldsymbol{v}_l(t)\boldsymbol{x}_j(t) & \text{Otherwise} \end{cases} \tag{51}$$

Since the term added at each step, $\boldsymbol{v}_l(t)\boldsymbol{x}_j(t)$, only depends on l and j, by introducing a matrix $\hat{\boldsymbol{K}}(t) \in \mathbb{R}^{N \times N}$ such that, for all $l, j \in \{1, ..., N\}$, $\hat{\boldsymbol{K}}_{l,j}(t) = \frac{\partial \boldsymbol{W}_{l,i}(t)}{\partial \boldsymbol{K}_{i,j}}$ for all $i \in \{1, ..., N\}$, we can compactly express the 4-dimensional sensitivity tensor with elements $\frac{\partial \boldsymbol{W}_{l,m}(t)}{\partial \boldsymbol{K}_{i,j}}$ using the 2-dimensional sensitivity matrix with elements $\hat{\boldsymbol{K}}_{l,j}(t)$ through sparsity and parameter-sharing obtained as directly consequences of the forward computation of the FWP defined above. Hence, the space complexity of RTRL is reduced to $O(N^2)$. Now Eq. 48 also greatly simplifies:

$$\frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{K}_{i,j}} = \sum_{l=1}^{N} \sum_{m=1}^{N} \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{W}_{l,m}(t)} \times \frac{\partial \boldsymbol{W}_{l,m}(t)}{\partial \boldsymbol{K}_{i,j}} = \sum_{l=1}^{N} \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{W}_{l,i}(t)} \times \frac{\partial \boldsymbol{W}_{l,i}(t)}{\partial \boldsymbol{K}_{i,j}} \tag{52}$$

By defining a matrix $\boldsymbol{E}(t) \in \mathbb{R}^{N \times N}$ such that $\boldsymbol{E}_{l,i}(t) = \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{W}_{l,i}(t)}$, Eq. 52 can be computed through one simple matrix multiplication; and Eq. 51 yields a simple forward recursion formula for $\hat{\boldsymbol{K}}(t)$:

$$\frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{K}} = \boldsymbol{E}(t)^{\intercal} \hat{\boldsymbol{K}}(t) \quad ; \quad \hat{\boldsymbol{K}}(t) = \hat{\boldsymbol{K}}(t-1) + \boldsymbol{v}(t) \otimes \boldsymbol{x}(t) \tag{53}$$

The time complexity of Eq. 53/Left (for one update) is $O(N^3)$ which is tractable. The batch version of Eq. 53/Left can be implemented as a *batch matrix-matrix multiplication* in standard deep learning libraries.

The derivation is analogous for $\boldsymbol{V}$ (by analogously defining $\hat{\boldsymbol{V}}(t)$) which yields:

$$\frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{V}} = \boldsymbol{E}(t) \hat{\boldsymbol{V}}(t) \quad ; \quad \hat{\boldsymbol{V}}(t) = \hat{\boldsymbol{V}}(t-1) + \boldsymbol{k}(t) \otimes \boldsymbol{x}(t) \tag{54}$$

In summary, RTRL for this simple FWP is tractable with the time/space complexities of $O(N^3)$ and $O(N^2)$. As for eLSTM (Sec. 5), this result is only valid for the one-layer case. Also note that the standard FWP (see e.g., [44]) applies softmax on both key and query vectors; here we need to remove such an activation function applied to the key (see Eq. 44) to make RTRL tractable.

A memory system view. The resulting system (consisting of the neural architecture, plus the RTRL learning algorithm) forms an interesting type of memory "organism." The system maintains two kinds of "synaptic" memories in an online fashion: the fast weight matrix $\boldsymbol{W}(t)$ is a short-term memory where the system stores information required to *solve the task* at hand (memory based on input observations; Eq. 44), while sensitivity matrices $\hat{\boldsymbol{K}}(t)$ and $\hat{\boldsymbol{V}}(t)$ (paired to the model's weight matrices $\boldsymbol{K}(t)$ and $\boldsymbol{V}(t)$) store another memory required for *learning* (memory based on external feedback to model outputs; Eqs. 53 and 54/Right).

Remarks. Strictly speaking, RTRL is the name of the learning algorithm when forward-mode automatic differentiation (AD) is applied to RNNs. Here we refer to RTRL as a generic name referring to the forward AD applied to any sequence models including linear Transformers. Regarding the standard Transformer: we note that a Transformer anyway needs to store all past activations even for its forward pass, and we are not aware of any straightforward "real-time learning algorithm" that leads to complexity reduction of any form.

A.3 Backpropagation Through Time (BPTT)

In Sec. 2, we review RTRL. For the sake of completeness, here we review BPTT using the same notations and settings described in Sec. 2.

BPTT can be obtained by directly summing derivatives of the *total loss* $\mathcal{L}^{\text{total}}(1,T)$ w.r.t. all intermediate variables $\boldsymbol{s}_k(t)$ for all $k \in \{1, ..., N\}$ and $t \in \{1, ..., T\}$:

$$\frac{\partial \mathcal{L}^{\text{total}}(1,T)}{\partial \boldsymbol{W}_{i,j}} = \sum_{t=1}^{T}\sum_{k=1}^{N} \frac{\partial \mathcal{L}^{\text{total}}(1,T)}{\partial \boldsymbol{s}_k(t)} \times \frac{\partial \boldsymbol{s}_k(t)}{\partial \boldsymbol{W}_{i,j}(t)} \tag{55}$$

where we introduce the notation $\frac{\partial \boldsymbol{s}_k(t)}{\partial \boldsymbol{W}_{i,j}(t)}$ denoting the derivative of $\boldsymbol{s}_k(t)$ w.r.t. the *variable* representing the weight $\boldsymbol{W}_{i,j}$ at time t, $\boldsymbol{W}_{i,j}(t)$ (meaning that $\boldsymbol{s}_k(t-1)$ is a constant w.r.t $\boldsymbol{W}_{i,j}(t)$ and thus, $\frac{\partial \boldsymbol{s}_k(t)}{\partial \boldsymbol{W}_{i,j}(t)}$ is only the first term of $\frac{\partial \boldsymbol{s}_k(t)}{\partial \boldsymbol{W}_{i,j}}$ in Eq. 4, i.e., $\frac{\partial \boldsymbol{s}_k(t)}{\partial \boldsymbol{W}_{i,j}(t)} = \boldsymbol{x}_j(t)\mathbb{1}_{k=i}$). This allows us to write gradients w.r.t. the weights at a specific time step, e.g., $\frac{\partial \mathcal{L}^{\text{total}}(1,T)}{\partial \boldsymbol{W}_{i,j}} = \sum_{\tau=1}^{T} \frac{\partial \mathcal{L}^{\text{total}}(1,T)}{\partial \boldsymbol{W}_{i,j}(\tau)}$.

By introducing the classic notation $\boldsymbol{\delta}(t) \in \mathbb{R}^N$ with $\boldsymbol{\delta}_k(t) = \frac{\partial \mathcal{L}^{\text{total}}(1,T)}{\partial \boldsymbol{s}_k(t)} \in \mathbb{R}$ for all $k \in \{1, ..., N\}$,

$$\boldsymbol{\delta}_k(t) = \sum_{\tau=1}^{T} \frac{\partial \mathcal{L}(\tau)}{\partial \boldsymbol{s}_k(t)} = \sum_{\tau=t}^{T} \frac{\partial \mathcal{L}(\tau)}{\partial \boldsymbol{s}_k(t)} = \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{s}_k(t)} + \sum_{\tau=t+1}^{T} \frac{\partial \mathcal{L}(\tau)}{\partial \boldsymbol{s}_k(t)} \tag{56}$$

$$= \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{s}_k(t)} + \sum_{n=1}^{N}\sum_{\tau=t+1}^{T} \frac{\partial \mathcal{L}(\tau)}{\partial \boldsymbol{s}_n(t+1)} \times \frac{\partial \boldsymbol{s}_n(t+1)}{\partial \boldsymbol{s}_k(t)} \tag{57}$$

Since $\sum_{\tau=t+1}^{T} \frac{\partial \mathcal{L}(\tau)}{\partial \boldsymbol{s}_n(t+1)} = \boldsymbol{\delta}_n(t+1)$ in Eq. 57, we obtain the backward recursion formula for $\boldsymbol{\delta}(t)$:

$$\boldsymbol{\delta}_k(t) = \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{s}_k(t)} + \sum_{n=1}^{N} \boldsymbol{\delta}_n(t+1) \times \frac{\partial \boldsymbol{s}_n(t+1)}{\partial \boldsymbol{s}_k(t)} = \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{s}_k(t)} + \sum_{n=1}^{N} \boldsymbol{\delta}_n(t+1)\boldsymbol{R}_{n,k}\sigma'(\boldsymbol{s}_k(t)) \tag{58}$$

$$\frac{\partial \mathcal{L}^{\text{total}}(1,T)}{\partial \boldsymbol{W}_{i,j}} = \sum_{t=1}^{T}\sum_{k=1}^{N} \boldsymbol{\delta}_k(t) \times \frac{\partial \boldsymbol{s}_k(t)}{\partial \boldsymbol{W}_{i,j}(t)} = \sum_{t=1}^{T} \boldsymbol{\delta}_i(t)\boldsymbol{x}_j(t) \tag{59}$$

In the end, all quantities involved in these equations can be expressed as matrix-matrix/vector multiplications, element-wise vector multiplications $\odot$, or outer products between two vectors $\otimes$:

$$\boldsymbol{\delta}(t) = \frac{\partial \mathcal{L}(t)}{\partial \boldsymbol{s}(t)} + (\boldsymbol{R}^{\intercal}\boldsymbol{\delta}(t+1)) \odot \sigma'(\boldsymbol{s}(t)) \quad ; \quad \frac{\partial \mathcal{L}^{\text{total}}(1,T)}{\partial \boldsymbol{W}} = \sum_{t=1}^{T} \boldsymbol{\delta}(t) \otimes \boldsymbol{x}(t) \tag{60}$$

The formula for $\frac{\partial \mathcal{L}^{\text{total}}(1,T)}{\partial \boldsymbol{R}}$ can be derived analogously. This is the derivation which is obtained as a "natural" extension of the standard backpropagation algorithm for feedforward networks, by unfolding the RNN over time. BPTT requires to store intermediate activations $\boldsymbol{s}(t) \in \mathbb{R}^N$ (and inputs $\boldsymbol{x}(t) \in \mathbb{R}^D$) for all $t \in \{1, .., T\}$, resulting in the space complexity of $O(T(N+D)) \sim O(TN)$. We can only compute the gradients $\frac{\partial \mathcal{L}^{\text{total}}(1,T)}{\partial \boldsymbol{W}}$ after going through the entire sequence (because of the backward recursion to compute $\boldsymbol{\delta}(t)$; Eq. 60/Left). The corresponding (per-update) time complexity is $O(TN^2)$.

Technically speaking, one could also adopt BPTT for online learning (by applying BPTT at each time step; *real-time BPTT* [47]). However, that would result in many redundant computations, with a time complexity of order of $O(T^2N^2)$ which is intractable in practice, and more expensive than RTRL's $O(N^4)$ for long sequences with $T > N$.

B Experimental Details and Extra Results

B.1 Diagnostic Tasks

Copy task. Here we provide details of the experiments on the copy task presented in Sec. 4.1. Let ℓ be a positive integer. Sequences in the copy task have a length 2ℓ where the ℓ first symbols are either 0 or 1, and all others are $\#$. The task is to sequentially read symbols in such a sequence, and to output the binary pattern presented in the first part of the sequence when reading the sequence of $\#$ in the second part. We conduct experiments with $\ell = \{50, 500\}$. Unlike prior work (see, e.g., [10]), we do not introduce any curriculum learning; we train models from scratch on the entire dataset containing sequences of length ranging from 2 to 2ℓ. Note that we do not conduct much of a hyper-parameter search; we find that for this task, trying a few values is enough to find configurations that achieve 100% accuracy. The corresponding hyper-parameters are as follows. Respectively for $\ell = (50, 500)$, we use a learning rate of $(1e^{-4}, 3e^{-5})$, a batch size of $(512, 128)$, and a hidden layer size of $(1024, 2048)$. We apply a gradient norm clipping of 1.0, and use the Adam optimiser. The model has no input embedding layer; one-hot representations of the input symbols are directly fed to the recurrent eLSTM layer (Sec. 3.1).

Remarks on computational capabilities of eLSTM. As noted in the introduction, eLSTM has certain limitations in terms of computational capabilities. In fact, certain theoretical results/limitations known for one-layer Quasi-RNN [48] directly apply to eLSTM (and other element-wise recurrent NNs). While we do not observe any empirical limitations on the main tasks of this work (Sec. 4), we find one algorithmic task on which eLSTM is not successful (at least we failed to find successful configurations), which is the code execution task [49, 44]. While we refer to the corresponding papers for the task description, this task requires the model to maintain dynamically changing values of a certain number of variables. We are not successful at finding any configuration of our eLSTM that achieves 100% sequence-level accuracy on this task, while this is straightforward with the standard fully recurrent LSTM.

Remarks on language modelling. Another popular "diagnostic" task used in recent RTRL research is small-scale language modelling (see, e.g., [10]). However, as we already discuss in Sec. 5, in practice, language modelling may not be the best target application of RTRL in modern deep learning (apart from test-time online adaptation, i.e., dynamic evaluation [50, 51], maybe). While we also conduct brief language modelling experiments using the Penn Treebank dataset (as in [10]) in our preliminary studies, we do not find any interesting result (beyond tiny perplexity improvements) that is worth being reported here.

B.2 DMLab

Here we provide details of our DMLab experiments. We use two memory tasks `rooms_select_nonmatching_object` and `rooms_watermaze` in Sec. 4.2, and one reactive task `room_keys_doors_puzzle` in Sec. 4.3. For the corresponding task descriptions and examples of game screenshots, we refer to `https://github.com/deepmind/lab/blob/master/game_scripts/levels/contributed/dmlab30/README.md#select-non-matching-object`. We use observations based on DMLab's `RGB_INTERLEAVED`. Our base model architecture is IMPALA's deep variant [31]. Action space discretisation is also based on IMPALA [31]. Note that, unlike our work, some prior work (e.g., R2D2 [32]) use some "improved" versions of the action space discretisation [52]. Regarding the reward clipping, no sophisticated asymmetric clipping [31] is used but the simple $[-1, 1]$ clipping. Training hyper-parameters/configurations are listed in Table 3. For the memory tasks, pre-training (Sec. 4.2) is done using TBPTT with $M = 100$. Our implementation is based on `torchbeast` [53]. We run three main training runs. Each run requires about a day of training on a V100 GPU (this is also the case with ProcGen and Atari). Also note that DMLab is CPU intensive. For evaluation, we first evaluate the final model checkpoint on three sets of 100 test episodes each; resulting in three mean test scores for each training run. We average these scores to obtain a single mean score for each training run. The final number we report is the mean and standard deviation of these scores across three training runs.

Table 3: Hyper-parameters for RL experiments. Parameters at the bottom are common to all settings (which are essentially taken from the Atari configuration of Espeholt et al. [31]).

Parameters	DMLab	Atari	Procgen
Input image dimension	3x96x72	1x84x84	3x64x64
4-frame stack	No	Yes	No
Action repeat	4	4	No
RNN dimension	512	256	256
Number of IMPALA actors		48	
Discount		0.99	
Learning rate		0.0006	
Batch size		32	
Gradient clipping		40	
Loss scaling factors (baseline, entropy)		(0.5, 0.01)	
RMSProp (alpha, epsilon, momentum)		(0.99, 0.01, 0)	

B.3 ProcGen

All our experimental settings in ProcGen enviroments are the same as for DMLab described above, except that no action repeat is used (see Table 3 for a comprehensible overview). Following [29], we use 500 levels for training. As we note in the main text, *Chaser* is the only ProcGen environment where we observe clear benefits of recurrent policies. In our preliminary studies, we also test *Caveflyer*/hard, *Caveflyer*/memory, *Dodgeball*/memory, *Jumper*/hard, *Jumper*/memory, and *Maze*/memory. However, in our settings, we do not observe any notable benefits of LSTM-based policies in these environments over the feedforward baseline, except some improvements in *Dodgeball*/memory mode: 10.1 ± 0.4 (feedforward) vs. 12.7 ± 0.6 (LSTM) on the training set.

B.4 Atari

All our experimental settings in the Atari environments are the same as for DMLab described above, except that 4-frame stacking is used (see also Table 3), and that we use 5 sets of 30 episodes each for evaluation. As stated in the main text, our selection of five environments follows Kapturowski et al. [32]. Regarding other tasks, we also consider *Solaris* from Atari, as Kapturowski et al. [32] find longer BPTT spans useful for this task. However, they achieve such improvements by training for 10 B environment frames, which are beyond our compute budget (800 M frames is our reasonable number); we tried up to 4 B steps, but without observing benefits of RTRL. Similarly, no real benefit of RTRL is observed on *Skiing* within this number of steps (except that RTRL and Feedforward agents immediately achieve a score around -8987 but remain stuck there forever, while the score of TBPTT-based agents gradually increases but only until circa -17418).

References

[1] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[2] Ronald J Williams and David Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection science*, 1(1):87–111, 1989.

[3] Anthony J Robinson and Frank Fallside. *The utility driven dynamic error propagation network*, volume 1. University of Cambridge Department of Engineering Cambridge, 1987.

[4] Anthony J Robinson. *Dynamic error propagation networks.* PhD thesis, University of Cambridge, 1989.

[5] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

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

[7] Ronald J Williams and Jing Peng. An efficient gradient-based algorithm for online training of recurrent network trajectories. *Neural computation*, 2(4):490–501, 1990.

[8] Corentin Tallec and Yann Ollivier. Unbiased online recurrent optimization. In *Int. Conf. on Learning Representations (ICLR)*, Vancouver, Canada, April 2018.

[9] Asier Mujika, Florian Meier, and Angelika Steger. Approximating real-time recurrent learning with random kronecker factors. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pages 6594–6603, Montréal, Canada, December 2018.

[10] Frederik Benzing, Marcelo Matheus Gauy, Asier Mujika, Anders Martinsson, and Angelika Steger. Optimal kronecker-sum approximation of real time recurrent learning. In *Proc. Int. Conf. on Machine Learning (ICML)*, volume 97, pages 604–613, Long Beach, CA, USA, June 2019.

[11] Jacob Menick, Erich Elsen, Utku Evci, Simon Osindero, Karen Simonyan, and Alex Graves. Practical real time recurrent learning with a sparse approximation. In *Int. Conf. on Learning Representations (ICLR)*, Virtual only, May 2021.

[12] David Silver, Anirudh Goyal, Ivo Danihelka, Matteo Hessel, and Hado van Hasselt. Learning by directional gradient descent. In *Int. Conf. on Learning Representations (ICLR)*, Virtual only, April 2022.

[13] James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. In *Int. Conf. on Learning Representations (ICLR)*, Toulon, France, April 2017.

[14] Tao Lei, Yu Zhang, Sida I. Wang, Hui Dai, and Yoav Artzi. Simple recurrent units for highly parallelizable recurrence. In *Proc. Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4470–4481, Brussels, Belgium, November 2018.

[15] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proc. Int. Conf. on Machine Learning (ICML)*, Virtual only, July 2020.

[16] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to recurrent nets. Technical Report FKI-147-91, Institut für Informatik, Technische Universität München, March 1991.

[17] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear Transformers are secretly fast weight programmers. In *Proc. Int. Conf. on Machine Learning (ICML)*, Virtual only, July 2021.

[18] Michael C. Mozer. A focused backpropagation algorithm for temporal pattern recognition. *Complex Systems*, 3(4):349–381, 1989.

[19] Michael C. Mozer. Induction of multiscale temporal structure. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 275–282, Denver, CO, USA, December 1991.

[20] Khurram Javed, Martha White, and Richard S. Sutton. Scalable online recurrent learning using columnar neural networks. *Preprint arXiv:2103.05787*, 2021.

[21] Khurram Javed, Haseeb Shah, Rich Sutton, and Martha White. Online real-time recurrent learning using sparse connections and selective learning. *Preprint arXiv:2302.05326*, 2023.

[22] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 1008–1014, Denver, CO, USA, November 1999.

[23] Richard S. Sutton, David A. McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 1057–1063, Denver, CO, USA, November 1999.

[24] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 1928–1937, New York City, NY, USA, June 2016.

[25] Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Recurrent policy gradients. *Logic Journal of IGPL*, 18(2):620–634, 2010.

[26] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[27] Jürgen Schmidhuber. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *Proc. Int. Joint Conf. on Neural Networks (IJCNN)*, pages 253–258, San Diego, CA, USA, June 1990.

[28] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *Preprint arXiv:1612.03801*, 2016.

[29] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 2048–2056, Virtual only, July 2020.

[30] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[31] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-RL with importance weighted actor-learner architectures. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 1406–1415, Stockholm, Sweden, July 2018.

[32] Steven Kapturowski, Georg Ostrovski, John Quan, Rémi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *Int. Conf. on Learning Representations (ICLR)*, New Orleans, LA, USA, May 2019.

[33] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[34] Barak A Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989.

[35] Michael Gherrity. A learning algorithm for analog, fully recurrent neural networks. In *Proc. Int. Joint Conf. on Neural Networks (IJCNN)*, volume 643, 1989.

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

[37] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper RNN. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 5457–5466, Salt Lake City, UT, USA, June 2018.

[38] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[39] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[40] Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, Matthew M. Botvinick, Nicolas Heess, and Raia Hadsell. Stabilizing Transformers for reinforcement learning. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 7487–7498, Virtual only, July 2020.

[41] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[42] Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *AAAI Fall Symposia*, pages 29–37, Arlington, VA, USA, November 2015.

[43] Alexander Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo Jimenez Rezende. Towards interpretable reinforcement learning using attention augmented agents. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pages 12329–12338, Vancouver, Canada, December 2019.

[44] Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Virtual only, December 2021.

[45] Kürt Meert and Jacques Ludik. A multilayer real-time recurrent learning algorithm for improved convergence. In *Proc. Int. Conf. on Artificial Neural Networks (ICANN)*, pages 445–450, Lausanne, Switzerland, October 1997.

[46] Jürgen Schmidhuber. A fixed size storage $o(n^3)$ time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, 4(2):243–248, 1992.

[47] Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures, and applications*, 433:17, 1995.

[48] William Merrill, Gail Weiss, Yoav Goldberg, Roy Schwartz, Noah A. Smith, and Eran Yahav. A formal hierarchy of RNN architectures. In *Proc. Association for Computational Linguistics (ACL)*, pages 443–459, Virtual only, July 2020.

[49] Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing some limitations of Transformers with feedback memory. *Preprint arXiv:2002.09402*, 2020.

[50] Tomás Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proc. Interspeech*, pages 1045–1048, Makuhari, Japan, September 2010.

[51] Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. Dynamic evaluation of neural sequence models. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 2771–2780, Stockholm, Sweden, July 2018.

[52] Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 3796–3803, Honolulu, HI, USA, January 2019.

[53] Heinrich Küttler, Nantas Nardelli, Thibaut Lavril, Marco Selvatici, Viswanath Sivakumar, Tim Rocktäschel, and Edward Grefenstette. Torchbeast: A PyTorch platform for distributed RL. *Preprint arXiv:1910.03552*, 2019.