A rapid and efficient learning rule for biological neural circuits

Eren Sezener[*1], Agnieszka Grabska-Barwińska[*1], Dimitar Kostadinov[*2], Maxime Beau[2], Sanjukta Krishnagopal[3], David Budden[1], Marcus Hutter[1], Joel Veness[1], Matthew Botvinick[1,3], Claudia Clopath[†4], Michael Häusser[†2], Peter E. Latham[†3]

[*†]Equal contribution
[1]DeepMind
[2]Wolfson Institute for Biomedical Research, University College London
[3]Gatsby Computational Neuroscience Unit, University College London
[4]Department of Bioengineering, Imperial College London

August 17, 2022

Abstract

The dominant view in neuroscience is that changes in synaptic weights underlie learning. It is unclear, however, how the brain is able to determine which synapses should change, and by how much. This uncertainty stands in sharp contrast to deep learning, where changes in weights are explicitly engineered to optimize performance. However, the main tool for that, backpropagation, has two problems. One is neuroscience related: it is not biologically plausible. The other is inherent: networks trained with this rule tend to forget old tasks when learning new ones. Here we introduce the Dendritic Gated Network (DGN), a variant of the Gated Linear Network, which offers a biologically plausible alternative to backpropagation. DGNs combine dendritic 'gating' (whereby interneurons target dendrites to shape neuronal responses) with local learning rules to yield provably efficient performance. They are significantly more data efficient than conventional artificial networks, and are highly resistant to forgetting. Consequently, they perform well on a variety of tasks, in some cases better than backpropagation. Importantly, DGNs have structural and functional similarities to the cerebellum, a link that we strengthen by using *in vivo* two-photon calcium imaging to show that single interneurons suppress activity in individual dendritic branches of Purkinje cells, a key feature of the model. Thus, DGNs leverage targeted dendritic inhibition and local learning – two features ubiquitous in the brain – to achieve fast and efficient learning.

1 Introduction

A hallmark of intelligent systems is their ability to learn. Humans, for instance, are capable of amazing feats – language acquisition and abstract reasoning being the most notable – and even fruit flies can learn simple reward associations [1, 2]. It is widely believed that this learning is implemented via synaptic plasticity. But which synapses should change in response to, say the appearance of a reward, and by how much? This is especially hard to answer in humans, who have about 10^{14} synapses, but it is hard even in fruit flies, which have about 10^7 – corresponding to 10 million adjustable parameters.

One answer to this question is known: introduce a loss function (a function that measures some aspect of performance, with higher performance corresponding to lower loss), compute the gradient of the loss with respect to the weights (find the direction in weight space that yields the largest improvement in performance), and change the weights in that direction. If the weight changes are not too large, this will, on average, reduce the loss, and so improve overall performance.

This approach has been amazingly successful in artificial neural networks, and has in fact driven the deep learning revolution [3]. However, the algorithm for computing the gradient in deep networks is not directly applicable to biological systems, as first pointed out by [4, 5] (see also recent reviews [6–8]). There are several reasons for this. First, to implement backpropagation [9–11], referred to simply as backprop, neurons would need to know their outgoing weights. Second, backprop requires two stages: a forward pass (for computation) and a backward pass (for learning). Moreover, in the backward pass an error signal must propagate from higher to lower areas, layer by layer (Fig. 1A), and during that backward pass information from the forward pass must remain in the neurons. However, biological neurons do not know their outgoing weights, and there is no evidence for a complicated, time-separated backward pass.

Backprop also leads to another problem, at least in standard deep learning setups: it adapts to the data it has seen most recently, so when learning a new task it forgets old ones [12]. This is known as catastrophic forgetting, and prevents networks trained with backprop to display the lifelong learning that comes so easily to essentially all organisms [13, 14].

Driven in part by the biological implausibility of backprop, there have been several proposals for architectures and learning rules that might be relevant to the brain. These include feedback alignment [15,16], creative use of dendrites [17,18], multiplexing [19], and methods in which the error signal is fed directly to each layer rather than propagating backwards from the output layer [20–28]. A particularly promising method that falls into the latter category is embodied in Gated Linear Networks [29,30]. These networks, which were motivated from a machine learning rather than a neuroscience perspective, have obtained state-of-the-art results in regression and denoising [31], contextual bandit optimization [32], and transfer learning [33].

In Gated Linear Networks (GLNs), the goal of every neuron, irrespective of its layer, is to predict the target output based on the input from the layer directly below it. This is very different from backprop, in which neurons in intermediate layers extract features that make it easier for subsequent layers to predict the target (compare Figs. 1A and B).

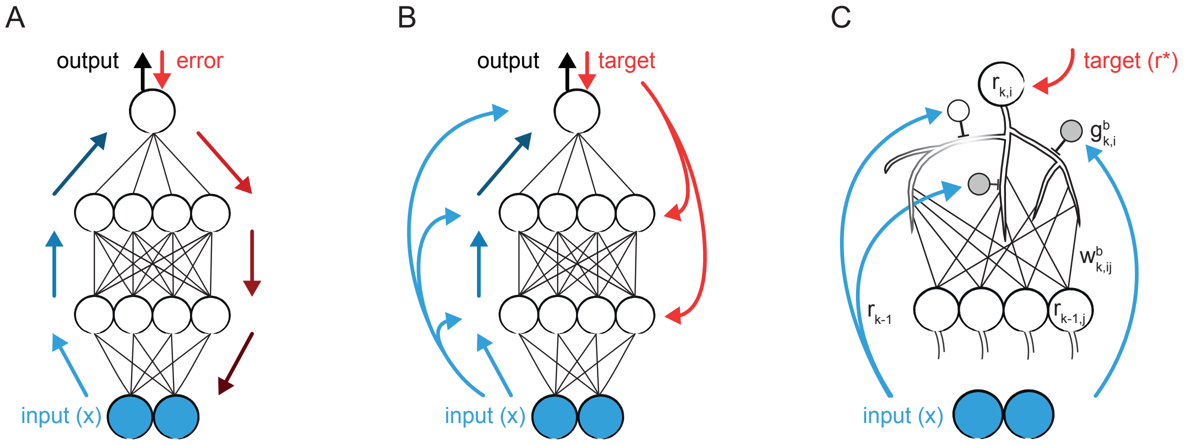

Figure 1: Comparison of multi-layer perceptrons (MLPs) and Dendrtic Gated Networks (DGNs). In all panels the blue filled circles at the bottom correspond to the input. **A**. MLP. Blue arrows show feedforward computations; red arrows show the error propagating back down. **B**. DGN. As with MLPs, information propagates up, as again shown by the blue arrows. However, rather than the error propagating down, each layer receives the target output, which it uses for learning. Connections from the input to each layer (light blue arrows) support the gating. **C**. A single postsynaptic neuron in layer k of a DGN, along with several presynaptic neurons in layer $k-1$. Each branch gets input from all the presynaptic neurons (although this is not necessary), and those branches are gated on and off by inhibitory interneurons which receive external input. The white interneuron is active, so its corresponding branch is gated off, as indicated by the light gray branches; the gray neurons are not active, so their branches are gated on.

Gated Linear Networks are thus particularly suitable for biologically plausible learning: every neuron is essentially part of a shallow network, with no hidden layers, for which the delta rule [34] – a rule that depends only on presynaptic and postsynaptic activity – is sufficient to learn.

To implement these local learning rules, the target activity (a scalar) is sent to every neuron, in every layer of the network (Fig. 1B, red arrows). This is typical of a large class of learning rules [20, 21, 23–27]. Completely atypical, though, is the role of the external input. It is used for gating the weights: each neuron has a bank of weights at its disposal, and the external input determines which one from that bank is used. For example, a neuron might use one set of weights when the visual input contains motion cues predominantly to the right; another set of weights when it contains motion cues predominantly to the left; and yet another when there are no motion cues at all. (Note that this example is over-simplified: in practice the input is high dimensional, and the mapping from external input to the chosen set of weights contains very little structure; see Fig. 2C.)

Having a “look-up” table, in which each input corresponds to a particular set of weights, is inconsistent with what we see in the brain. However, we can attain the performance of Gated Linear networks by gating dendritic branches on and off, using

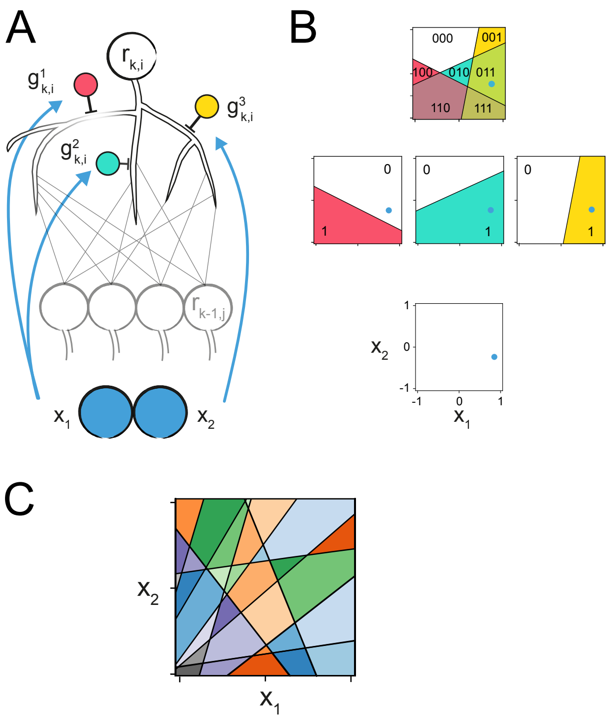

Figure 2: Random mapping (here implemented with so-called half-space gating, which we use throughout the paper; see Eq. (3)), shown for two dimensional input for clarity. (In realistic cases the input is, of course, high dimensional.) **A**. Two input neurons (blue) connect to three dendritic branches, and gate them either on or off. Each gate divides the two dimensional input into two half-spaces, one of which shuts down the gate, silencing the corresponding dendritic branch. **B**. For the input on this particular trial (blue dot in bottom square), the gate $g^1_{k,i}$ (red) is off, while $g^2_{k,i}$ (teal) and $g^3_{k,i}$ (yellow) are on, as indicated by the three squares with the corresponding colors. The top square shows a summary of the possible combinations of weights. Each of the seven regions has a different combination, making it possible to implement a large range of input-output mappings. **C**. A more realistic case of 10 branches. In DGNs, each coloured region corresponds to a linear combination of 10 sets of weights. This is in contrast to GLNs, which use a separate set of weights for each region.

inhibitory neurons, in an input-dependent manner (Figs 1C and 2). We thus replace the weight look-up mechanism of GLNs with linearly additive dendritic weights, and refer to these networks as Dendritic Gated Networks (DGNs). Perhaps surprisingly, the mapping from the input to the dendritic branches is completely random, so the input isn't chosen to target specific branches (see, for example, Fig. 2C). This stands in sharp contrast to architectures where the gating is learned [35]. But the unlearned, random mapping is in fact a key ingredient, as it allows DGNs to represent essentially arbitrary nonlinear functions efficiently. Moreover, this gating makes DGNs especially resistant to forgetting. In particular, when data comes in separate "tasks", DGNs can learn new ones without forgetting the old. Finally, the loss is a convex function of the weights for each unit (see Supplementary Information, "Convexity"), as it is in Gated Linear Networks [29]. Convexity is an extremely useful feature, as it enables DGNs, like the Gated Linear Networks on which they are based, to learn quickly.

Below we describe multi-layer Dendritic Gated Networks in detail – both the architecture and the learning rule. We then train them on four tasks: two on which vanilla feedforward networks trained with backprop typically exhibit catastrophic forgetting, and two relevant to the cerebellum. We map the proposed learning rule and the associated architecture to the cerebellum because 1) the climbing fibers provide a well-defined feedback signal; 2) its input-output function is relatively linear [36–38]; and 3) molecular layer interneurons could act as gates [39–48]. Finally, we demonstrate experimentally that a key prediction of the DGN – suppression of individual dendritic branches by interneurons – is observed in cerebellar Purkinje cell spiny branchlets *in vivo.* Thus, our theoretical and experimental results draw a specific link between learning in DGNs and the functional architecture of the cerebellum. The generality of the DGN architecture should also allow this algorithm to be implemented in a range of networks in the mammalian brain, including the neocortex.

2 Results

2.1 Dendritic Gated Networks

Dendritic Gated Networks, like conventional deep networks, are made up of multiple layers, with the input to each layer consisting of a linear combination of the activity in the previous layer. Unlike conventional deep networks, however, the weights are controlled by external input, via gating functions, denoted $g(\mathbf{x})$; those functions are implemented via dendritic branches (Figs. 1B, C and 2). This results in the following network equations. The activity (i.e., the instantaneous firing rate) of the i^{th} neuron in layer k, denoted $r_{k,i}$, is

$$r_{k,i} = \phi\left(\sum_{b=1}^{B_{k,i}} g_{k,i}^{b}(\mathbf{x}) \sum_{j=0}^{n_{k-1}} w_{k,ij}^{b} h_{k-1,j}\right), \tag{1}$$

with the synaptic drive, $h_{k-1,j}$, given in terms of $r_{k-1,j}$ as

$$h_{k-1,j} = \phi^{-1}(r_{k-1,j}) \,. \tag{2}$$

Here $\phi(\cdot)$ is the activation function (either identity or sigmoid), $r_{k,i}$ is the activity of i^{th} neuron in layer k (with $r_{k,0}$ set to 1 to allow for a bias), $B_{k,i}$ is the number of branches of neuron i in layer k, $w^b_{k,ij}$ is the weight from neuron j in layer $k-1$ to the b^{th} branch of neuron i in layer k, n_k is the number of neurons in layer k, and $g^b_{k,i}(\mathbf{x})$ is the binary gating variable; depending on the external input, $\mathbf{x}$ (taken to be an n-dimensional vector), it's either 1 (in which case the b^{th} branch of the i^{th} neuron is gated on) or 0 (in which case it's gated off). There are K layers, so k runs from 1 to K. The input to the bottom layer is $\mathbf{x}$ – the same as the input to the gating variable.

The mapping from the input, $\mathbf{x}$, to the gating variable, $g^b_{k,i}(\mathbf{x})$, is not learned; instead, it is pre-specified, and does not change with time. In all of our simulations we use random half-space gating [29]; that is,

$$g^b_{k,i}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{v}^b_{k,i} \cdot \mathbf{x} \geq \theta^b_{k,i} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where $\mathbf{v}^b_{k,i}$ and $\theta^b_{k,i}$ are sampled randomly and kept fixed throughout learning (see Methods), and "$\cdot$" is the standard dot product.

Note that a Dendritic Gated Network with one branch reduces to a Gated Linear Network. With a caveat: in the original formulation [29], Gated Linear Networks had a nonzero weight for each input, $\mathbf{x}$, which is not the case if weights are completely gated off for one of the half spaces (because in that case the weights are zero). For a detailed description of the difference between GLNs and DGNs, see Supplementary Information, "Difference between GLNs and DGNs".

In Dendritic Gated Networks, the goal of each neuron is to predict the target output, denoted r^* (which is a function of $\mathbf{x}$; we suppress the $\mathbf{x}$-dependence to reduce clutter). To do that, the weights, $w^b_{k,ij}$, are modified to reduce the loss, $\ell_k(r^*, r_{k,i})$. For weight updates we use gradient descent,

$$\Delta w^b_{k,ij} = -\eta \, \frac{\partial \ell(r^*, r_{k,i})}{\partial w^b_{k,ij}} \tag{4}$$

where η is the learning rate, and the updates are performed after each sample. The form of the loss can influence both the speed of learning and the asymptotic performance, but conceptually we should just think of it as some distance between r^* and $r_{k,i}$. In the simplest case, which is suitable for regression, ϕ is the identity ($r_{k,i} = h_{k,i}$) and the loss is quadratic,

$$\ell(r^*, r_{k,i}) = \frac{1}{2}(r^* - r_{k,i})^2 \,, \tag{5}$$

so the update rule is

$$\Delta w^b_{k,ij} = \eta \, g^b_{k,i}(\mathbf{x})(r^* - r_{k,i}) \, h_{k-1,j} \,. \tag{6}$$

This has the form of a gated version of the delta rule [34]. For classification, a different loss function is more appropriate. However, the update rule still has the form of a gated version of the delta rule; see Methods, Sec. 4.1, for details.

2.2 Simulations

Equations (1) and (3) for the network dynamics and Eq. (4) for learning constitute a complete description of our model. For a given problem, we just need to choose a target input-output relationship (a mapping from $\mathbf{x}$ to r^*) and specify the loss functions, $\ell(r^*, r_{k,i})$. Here we consider four tasks. The first two, designed to illustrate the resistance of DGNs to catastrophic forgetting, are classification tasks, for which we use a sigmoid activation and cross-entropy loss (Methods, Sec. 4.1); the second two, which are relevant to the cerebellum, are regression tasks, for which we use an identity activation and quadratic loss, as just described.

DGNs can mitigate catastrophic forgetting.

Animals are able to acquire new skills throughout life, seemingly without compromising their ability to solve previously learned tasks [13, 14]. Standard networks do not share this ability: when trained on two tasks in a row, they tend to forget the first one. This phenomenon, known as "catastrophic forgetting", is an old problem [49–51], and many algorithms have been developed to address it. These typically fall into two categories. The first involves replaying tasks previously seen during training [51–53]. The second involves explicitly maintaining additional sets of model parameters related to previously learned tasks. Examples include freezing a subset of weights [54, 55], dynamically adjusting learning rates [56], and augmenting the loss with regularization terms with respect to past parameters [57–59]. A limitation of these approaches (aside from additional algorithmic and computational complexity) is that they require task boundaries to be provided or accurately inferred.

Unlike contemporary neural networks, the DGN architecture and learning rule is naturally robust to catastrophic forgetting without any modifications or knowledge of task boundaries (something that has been shown for Gated Linear Networks as well [30]). In Fig. 3 we illustrate, on a simple task, the mechanism behind this robustness, and show how it differs from a standard multi-layer perceptron; details are given in the caption.

To demonstrate robustness to catastrophic forgetting on a more challenging task, we train a DGN on the pixel-permuted MNIST continual learning benchmark [57, 60]. In this benchmark, the network has to learn random permutations of the input pixels, with the random permutation changing every 60,000 trials (see Methods for additional details). We compare the DGN to a multi layer perceptron (MLP) with and without elastic weight consolidation (EWC) [57], the latter a highly-effective method explicitly designed to prevent catastrophic forgetting by storing parameters of previously seen tasks. Although elastic weight consolidation is effective, it requires a very complicated architecture. In addition, it must be supplied with task boundaries, so it receives more information than the DGN.

Because MNIST has 10 digits, we train 10 different DGNs. This could be reduced to 4 networks (in general $\log_2$ of the number of outputs) by using a more efficient code – one in which each network divides the 10 digits into two classes. Alternatively, we could use a single DGN where each unit has a 10 dimensional output corresponding to the class probabilities. However, this is not biologically plausible, so we did not use it.

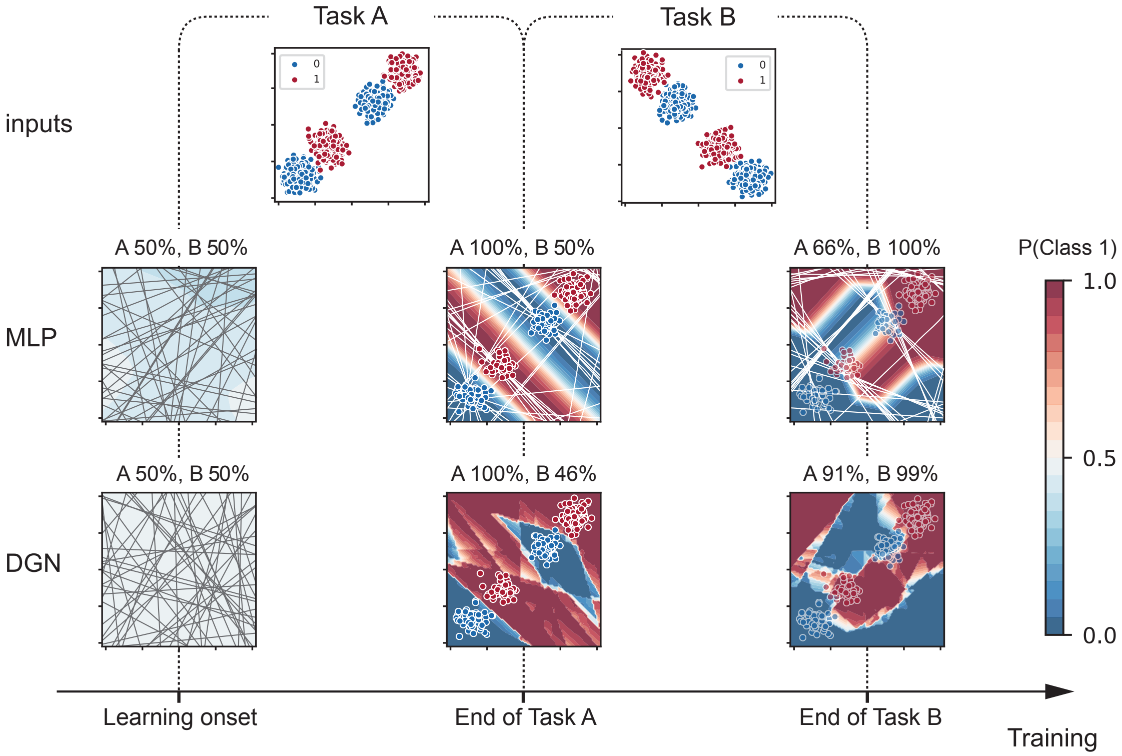

Figure 3: Comparison of the DGN to a standard multi-layer perceptron (MLP) trained with backprop. Each point on the square has to be classified as "blue" (class 0) or "red" (class 1). We consider a scenario common in the real world (but difficult for standard networks): the data comes in two separate tasks, as shown in the first row. We trained a 2-layer MLP (second row) and a 2-layer DGN (third row) on the two tasks. The output of the network is the probability of each class, as indicated by color; the percentages report the accuracy for each of the tasks. The MLP uses ReLU activation functions, so each neuron has an effective gating; the boundaries of those gates are shown in gray. The boundaries move with learning, and are plotted at the end of training of each of the tasks (white lines). The boundaries of the DGN do not move, so we plot them only in the first column. After training on Task A, most of the boundaries in the MLP are aligned at -45 degrees, parallel to the decision boundaries, which allows the network to perfectly separate the two classes. In the DGN, the boundaries do not change, but the network also perfectly separates the two classes. However, after training on Task B, the DGN retains high performance on Task A (91%), while the MLP's performance drops to 66%. That's because many of the boundaries changed to the orthogonal direction (45 degrees). For the DGN, on the other hand, changes to the network were much more local, allowing it to retain the memory of the old task (see samples from Task A overlaid on all panels) while accommodating the new one. The MLP has 50 neurons in the hidden layer; the DGN has 5 neurons, each with 10 dendritic branches, in the hidden layer.

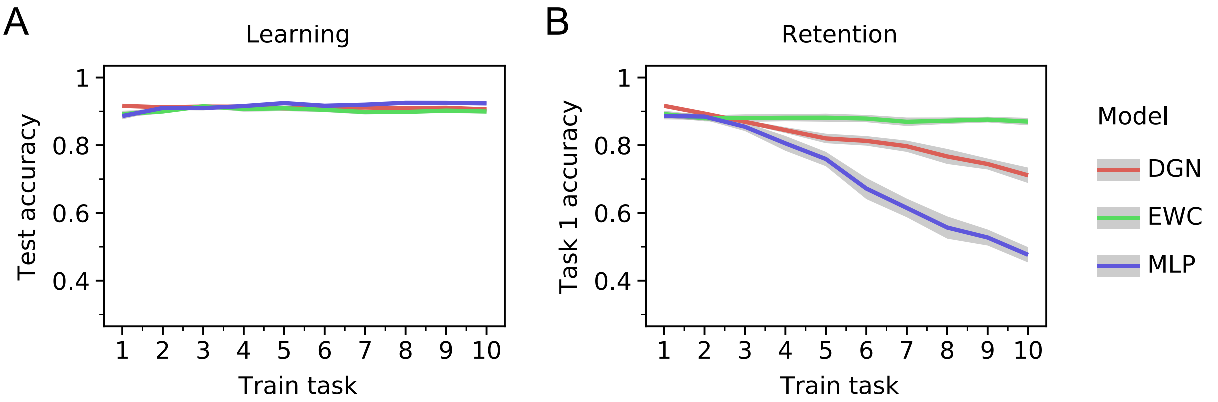

Figure 4: Learning and retention on the permuted MNIST task. The tasks are learned sequentially in a continual learning setup. **A**. Performance (on test data) for each of the 10 tasks, where a "task" corresponds to a random permutation of the pixels. **B**. Performance on the first task after each of nine new tasks is learned. As discussed in the main text, the MLP is especially bad at this task. The EWC is much better, to a large extent because it received extra information: the task boundaries. Even though the DGN was not given that information, it forgets a factor of two more slowly than the MLP. Error bars in both plots denote 95% confidence intervals over 20 random seeds.

Each of the 10 networks contains 3 layers, with 100, 20, and 1 neuron per layer, and there are 10 dendritic branches per neuron. The targets are categorical (1 if the digit is present, 0 if it is not), so we use binary cross-entropy rather than quadratic loss (see Methods, Sec. 4.1). We use 1000, 200, and 10 neurons per layer for the MLP (so that the number of weights match, approximately, the number weights in the DGN), with categorical cross entropy loss, both with and without elastic weight consolidation, and optimize the learning rates separately for each network.

Figure 4 shows the learning and retention performance of the DGN, with the MLP and EWC networks included primarily as benchmarks (neither is biologically plausible). In Fig. 4A we plot performance on each task for the three networks; as can be seen, performance is virtually identical. In Fig. 4B we investigate resistance to forgetting, by plotting the performance on the first task as the nine subsequent tasks are learned. The EWC network retains its original performance almost perfectly, the MLP forgets rapidly, and the DGN is in-between. It is not surprising that the EWC does well, as it was tailored to this task, and in particular it was explicitly given task boundaries. Somewhat more surprising is the performance of the DGN, which had none of these advantages but still forgets much more slowly than the MLP. The DGN also learns new tasks more rapidly than either the EWC or MLP networks (Supplementary Figure S3), because the loss is convex and learning is local.

Mapping DGNs to the Cerebellum

For the next two simulations we consider computations that can be mapped onto cerebellar circuitry. We focus on the cerebellum for several reasons: it is highly ex-

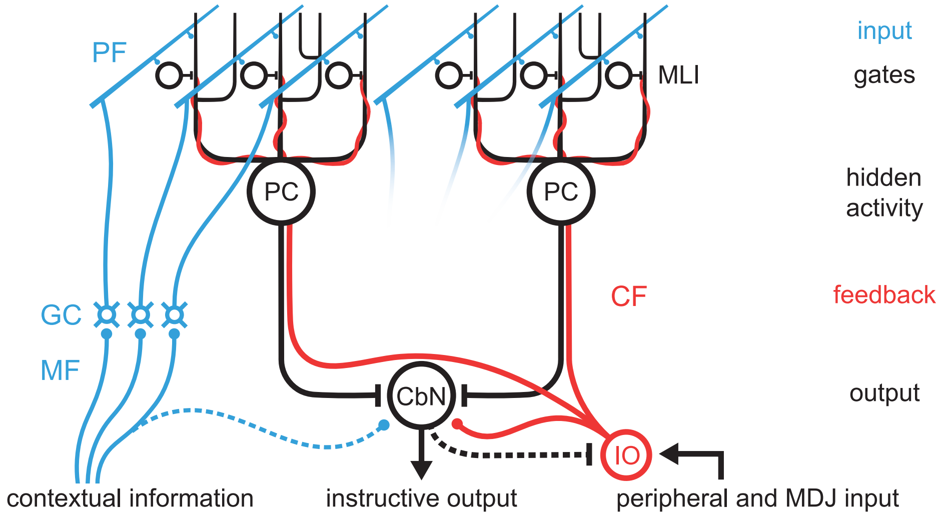

Figure 5: **The cerebellum as a two layer DGN.** Contextual information from the mossy fiber/granule cell (MF/GC) pathway is conveyed as input to the network via parallel fibers (PFs) that form synapses onto both the dendritic branches of Purkinje cells and molecular layer interneurons (MLIs). The inhibitory MLIs act as input-dependent gates of Purkinje cell dendritic branches. Purkinje cells converge onto the cerebellar nuclear neurons (CbNs), which constitute the output of the cerebellar network. The climbing fibers (CFs, red) originating in the inferior olive (IO) convey the feedback signal that is used to tune both the Purkinje cells, based on which inputs are gated on or off, and also the CbNs. Excitatory and inhibitory connections are depicted as round- and T-ends, respectively. Dashed lines represent connections not included in the model.

perimentally accessible; its architecture is well characterized; there is a clear feedback signal to the Purkinje cells (the cerebellar neurons principally involved in learning); its input-output function is relatively linear [36–38]; and molecular layer interneurons play a major role in shaping Purkinje cell responses [39–45, 47], and can influence climbing fiber-mediated dendritic calcium signals in Purkinje cells [46, 48, 61].

Both classic and more modern theoretical studies in the cerebellum have focused on the cerebellar cortex, modelling it as a one-layer feedforward network [62–66]. In this view, the parallel fibers project to Purkinje cells, and their synaptic weights are adjusted under the feedback signal from the climbing fibers. This picture, however, is an over-simplification, as Purkinje cells do not directly influence downstream structures. Instead, they project to the cerebellar nucleus, which constitutes the output of the cerebellum (see Fig. 5). The fact that Purkinje cells form a hidden layer, combined with the observed plasticity in the Purkinje cell to cerebellar nucleus synapses [67–71], means most learning rules tailored to one-layer networks, including the delta rule, cannot be used to train the network.

We propose instead that the cerebellum acts as a two layer DGN comprised of Purkinje cells as the first, hidden layer and the cerebellar nucleus as the second, output layer (Fig. 5). Parallel fibers provide the input to both the input layer (Purkinje cells) and the gates, represented by molecular layer interneurons, that control learning in individual Purkinje cell dendrites. For the output layer of the DGN (which consists of one neuron), we use a non-gated rather than a gated neuron, as the unique biophysical features of cerebellar nuclear neurons allow them to integrate inputs approximately linearly [72]. The climbing fibers provide the feedback signal to Purkinje cells and cerebellar nuclear neurons. In our formulation, climbing fiber feedback signals the target, allowing each neuron to compute its own local error by comparing the target to its output ($r_{k,i}$). This formulation is a departure from the strict error-coding role that is traditionally attributed to climbing fibers, but is consistent with a growing body of evidence that climbing fibers signal a variety of sensorimotor and cognitive predictions [73].

DGNs can learn inverse kinematics

The cerebellum is thought to implement inverse motor control [74, 75]. We therefore applied our proposed DGN network to the SARCOS benchmark [76], which is an inverse kinematics dataset from a 7 degree-of-freedom robot arm (Fig. 6). The goal is to learn an inverse model, and predict 7 torques given the joint positions, velocities, and accelerations for each of the 7 joints (corresponding to a 21 dimensional input).

The target output, r^*, is the desired torque, given the 21-dimensional input. There are seven joints, so we train seven different networks, each with its own target output. We use DGN networks with 20 Purkinje cells, and minimize a quadratic loss (5). Since this is a relatively hard task, performance depends strongly on the number of branches. In Fig. 6 we plot the target torques for each joint (dots) along with the predictions of the DGN (lines; chosen for ease of comparison as there is no data between the points) for 500 branches. The lines follow the points reasonably closely, even when there are large fluctuations, indicating that the DGN is faithfully predicting torques. The performance of our network (mean squared error on test data in the original torque units) is comparable to that of most existing machine learning algorithms (Supplementary Table S1) while using fewer samples to learn. In Supplementary Fig. S2 we show the equivalent plot for 5, 50 and 5000 branches. Even at 5 branches performance is reasonable, while at 5000 we exceed the performance of almost all existing machine learning algorithms.

Vestibulo-ocular reflex, and adaptation to gain changes

To maintain a stable image on the retina during head movements, when an animal moves its head it moves its eyes in the opposite direction. This is known as the vestibulo-ocular reflex (VOR), and a key feature of it is that it's plastic: animals can adapt quickly when the relationship between the head movement and visual feedback is changed, as occurs as animals grow or are given corrective lenses. VOR gain adaptation relies critically on the cerebellum, and has been used to study cerebellar motor learning for decades [77–81]. This is an easy task to learn – almost any network, including a linear one, can achieve high performance on it. We consider it primarily because it is a very common cerebellar task.

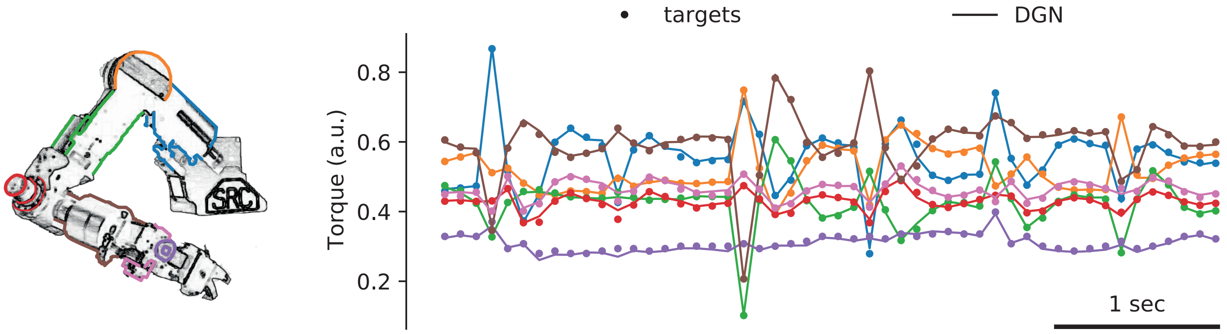

Figure 6: Sarcos experiment. DGNs can solve a challenging motor control task: predicting torques from the proprioceptive input. The data comes from a SARCOS dexterous robotic arm [76], pictured on the left. The inputs are position, velocity and acceleration of the 7 joints (a 21 dimensional variable); the targets are the desired torques (7 dimensional). Example targets (normalized to keep the training data between 0 and 1) are shown with dots, the lines are the output of our network. Performance is very good; only rarely is there a visible difference between the dots and the lines.

We applied our DGN network to the VOR in a regime where the gain occasionally changes abruptly. The gain, denoted G, is the ratio of the desired eye velocity to the head velocity (multiplied by -1 because the eyes and head move in opposite direction, to keep with the convention that the gain is reported as a positive number). When the gain is (artificially) changed, at first animals move their eyes at the wrong speed, but after about 15 minutes they learn to compensate [79, 80].

We trained our network on a head velocity signal of the form

$$s(t) = \sin(\omega_1 t) + \sin(\omega_2 t)\,, \tag{7}$$

with $\omega_1 = 13.333$ and $\omega_2 = 20.733$ (corresponding to 2.12 and 3.30 Hz, respectively). This signal was chosen to mimic, approximately, the irregular head velocities encountered in natural viewing conditions. Following Clopath et. al. [82], we assumed that the Purkinje cells receive delayed versions of this signal. The i^{th} input signal, $x_i(t)$, which arrives via the parallel fibers, is modelled as

$$x_i(t) = s(t - \tau_i)\,, \tag{8}$$

with delays, τ_i, spanning the range 50-300 ms. The cerebellum needs to compute the scaled version of the eye velocity: $r^*(t) = Gs(t)$ (as mentioned above, the actual eye movement is $-r^*(t)$, but we follow the standard convention). Learning was online, and we updated the weights every 500 ms, to approximately match the climbing fiber firing rate [83].

The DGN contained 20 Purkinje cells, with 10 branches each; these project to one output neuron (corresponding to the cerebellar nucleus), which was linear and not gated. As a baseline, we trained an MLP with the same number of weights (resulting in 200 hidden neurons). We used quadratic loss for both the DGN and the MLP and, as in [82], we assumed $n = 100$ parallel fibers and a single output. Each branch received input from all 100 parallel fibers. Gating (Eq. (3)) was controlled by $x_i(t)$ (given in Eq. (8)), reflecting the parallel fiber influence on molecular layer interneurons (Fig. 5); see Methods for details. Given the timescale of the signal (2-3 Hz), any individual branch was gated on for about 500 ms at a time. The networks were pre-trained on a gain, G, of 1. We implemented four jump changes: first to 0.7, then back to 1.0, then to 1.3, and, finally, back to 1.0; in all cases, for 30 minutes at each gain (Fig. 7A).

Performance for both the DGN and the MLP were comparable and, after suitably adjusting the learning rates, the networks were able to learn in 15-20 minutes (Fig. 7A, B), consistent with learning times in behavioral experiments [79, 80]. Figure 7C shows the target and predicted head velocities immediately before and after each gain change. Not surprisingly, immediately after a gain change, the network produces output with the old gain.

Figure 7D shows the connection strengths between parallel fibers ($x_i(t)$, Eq. (8)) and Purkinje cells, after learning, as a function of the delay, τ_i. There are two notable features to these plots. First, the connectivity patterns are smooth. Second, although the DGN and the MLP solve the task equally well, there is a clear difference: for the MLP the connectivity patterns are highly stereotyped, while for the DGN they are far less so.

The smooth connectivity patterns, which are seen in both MLPs and DGNs, arise primarily because weights mediating inputs with similar delays have similar updates during learning. But there is another, somewhat technical, reason: the weights were initialized to small values. That's important because for most directions in weight space, changes in the weights have no effect on the loss. Component of the weights that lie in these "null" directions will, therefore, not change with learning. Small initial weights ensure that the components in the null directions start small, and the lack of learning in these directions means they stay small.

The difference in the connectivity patterns – stereotyped versus diverse – are due to the fact that MLPs are not gated whereas DGNs are. The smooth, stereotyped connectivity patterns seen in MLPs arise because all neurons receive similar input statistics, and so they find similar solutions. The more diverse connectivity patterns seen in DGNs arise because inputs to different branches are gated differently, and so different branches do not see the same input statistics.

What happens when the initial weights are large and initially random? In that case, because the weights don't change in the null directions, the final connectivity patterns are also not smooth – they're almost as noisy as the initial weights. Here as well, though, there are difference between DGNs and MLPs: for DGNs the noise rides on top of diverse connectivity patterns very similar to those in the top panels of Fig. 7D, while for MLPs the noise is unmodulated, and more or less white (see Supplementary Fig. S4).

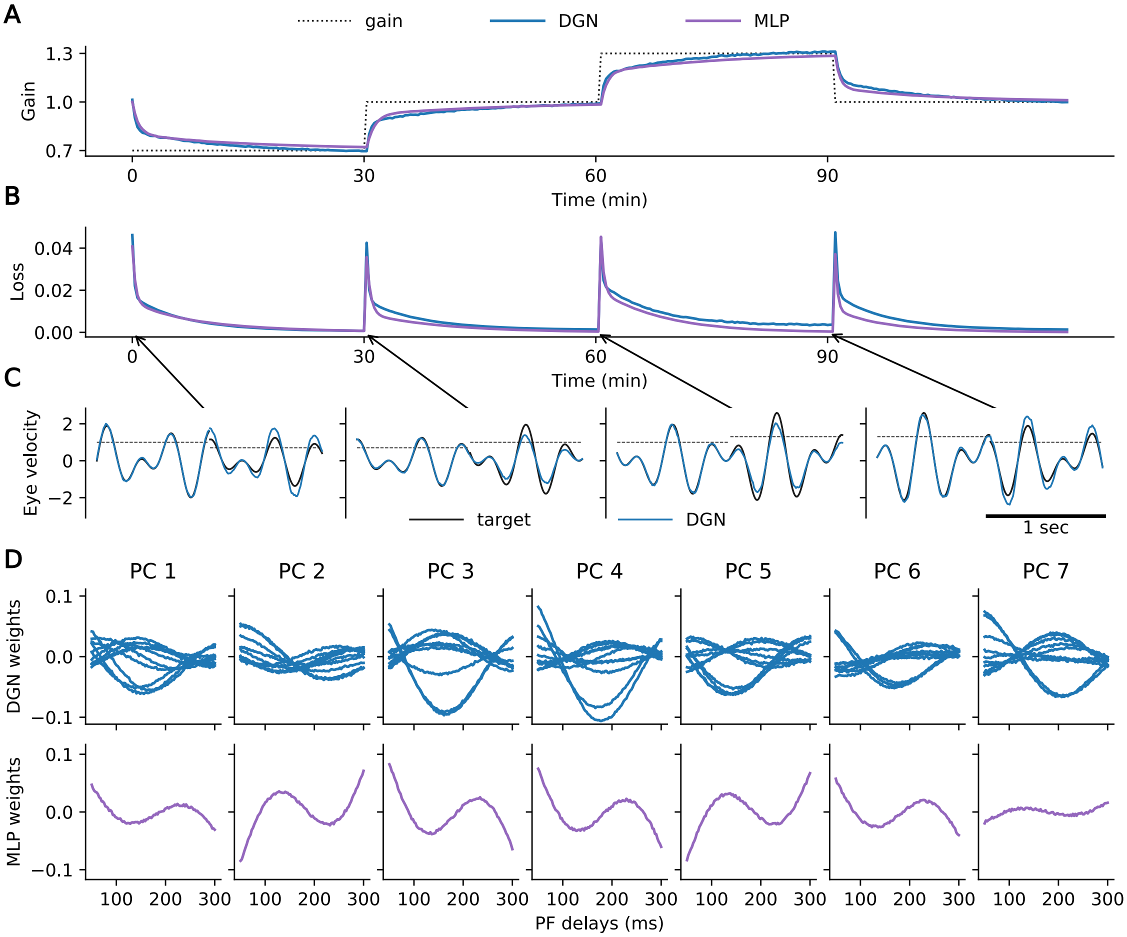

Figure 7: VOR adaptation task. We trained the networks on gain $G = 1$, then changed the gain every 30 minutes. Results are shown for the Dendritic Gated Network (DGN) and a multi-layer perceptron (MLP). **A**. Dashed lines are true gain versus time; blue and purple lines are gains computed by the DGN and MLP, respectively. For both networks, gains were inferred almost perfectly after 15-20 minutes. **B**. Performance, measured as mean squared error between the true angular velocity, $Gs(t)$ (Eq. (7)), and the angular velocity inferred by the networks. Same color code as panel A. **C**. Comparison of target angular velocity versus time (black) to that predicted by the DGN (blue). (A plot for the MLP is similar.) Before the gain change, the two are almost identical; immediately after the gain change, the network uses the previous gain. **D**. Top panel: Parallel fiber weights for the DGN network versus delay, τ_i (Eq. (8)). Each panel shows 10 branches; 5 Purkinje cells are shown (chosen randomly out of 20). The weights vary smoothly with delay. Bottom panel: MLP weight profile, except that dendritic branches are replaced by the whole neuron (all 100 parallel fibers). The weights again vary smoothly with delay, but their shapes are now highly stereotyped.

2.3 Testing predictions of the DGN in behaving animals.

A key feature of DGNs is that the gates should exert a local effect on the dendritic branches of the principal neurons. When mapped onto the cerebellum, this suggests that the molecular layer interneurons (MLIs) should inhibit the individual dendritic branches of the Purkinje cells. We therefore tested whether individual Purkinje cell dendrites can be inhibited by activity in MLIs. Previous *in vitro* work has demonstrated that synaptic inhibition can locally inhibit calcium signals in Purkinje cell dendrites [46], and *in vivo* work has shown that MLI activity can influence the variability of these signals [48, 61], but it has not yet been shown whether such effects can be localized to individual dendrites of Purkinje cells, and if so, what the spatial relationship of this effect is with presynaptic interneurons. Using multi-plane 2-photon imaging in awake PV-cre mice injected with the GCaMP7f virus, we could reliably record calcium signals from individual MLI somata and the climbing fiber-evoked calcium signals in the dendritic tree of Purkinje cells (Fig. 8A-B). With this approach, we were able to identify a substantial proportion of interneurons (72/142 MLIs in 3 mice; 51%) whose activation was associated with a significant decrease in climbing-fiber driven calcium signals in the dendrites of at least one nearby Purkinje cell(Fig. S5A). Given the axonal spread of MLIs [84,85], the analysis was confined to single interneurons that were located within 150 um (rostrocaudally) and 50 um (mediolaterally) from a given Purkinje cell dendrite. The extent of suppression varied between PC dendrites and also within individual PC dendrites recorded at different depths (Fig. 8C). In modulated Purkinje cell dendrites, the degree of suppression was 17.4 ± 0.5% ($n = 133$ Purkinje cells in 3 mice [range 6.6 to 53.5%]).

After identifying Purkinje cells whose global dendritic signals were inhibited when nearby MLIs were active, we investigated the spatial extent of this inhibition within Purkinje cell dendritic segments. We generated climbing fiber-evoked calcium signal maps in Purkinje cell dendrites, both when MLIs were active and when they were inactive (Fig. 8D, left). The difference revealed that MLI activation was associated with local suppression of calcium signals in subregions of the dendrites (blue region in Fig. 8E, left; Fig. S5B-E). To quantify the spatial extent of the suppression, we subdivided Purkinje cell dendritic regions into 1 μm segments to generate spatial activity profiles in MLI-active and MLI-inactive conditions (Figs. 8D, right). Subtracting these yielded a spatial difference trace (Figs. 8E, right). Aligning these segments across all modulated PC dendrites allowed us to determine the average spatial profile of suppressed dendritic segments (Figs. 8F). To identify false positives, for each Purkinje cell dendrite analyzed we generated a shuffled difference trace, where MLI-active and MLI-inactive traces were replaced with an equal mixture of the two conditions (odd-even event split). We then identified suppressed regions in these shuffled difference traces, and used the 95th percentile (4.6 μm) as the minimum length for a segment to be considered significantly modulated. Across all experiments, we identified $n = 77$ significantly modulated segments ($N = 3$ mice) whose calcium signals were suppressed by $42 \pm 2\%$ (mean ± S.E.M.) (Fig. 8G). The spatial extent of MLI-gated inhibition in these dendritic segments was 31 ± 3 μm (mean ± S.E.M.), accounting for $35 \pm 3\%$ (mean ± S.E.M.) of the extent of the dendritic tree at the imaged plane (Fig. 8H-I). These results demonstrate that MLIs can locally inhibit climbing fiber signals in the

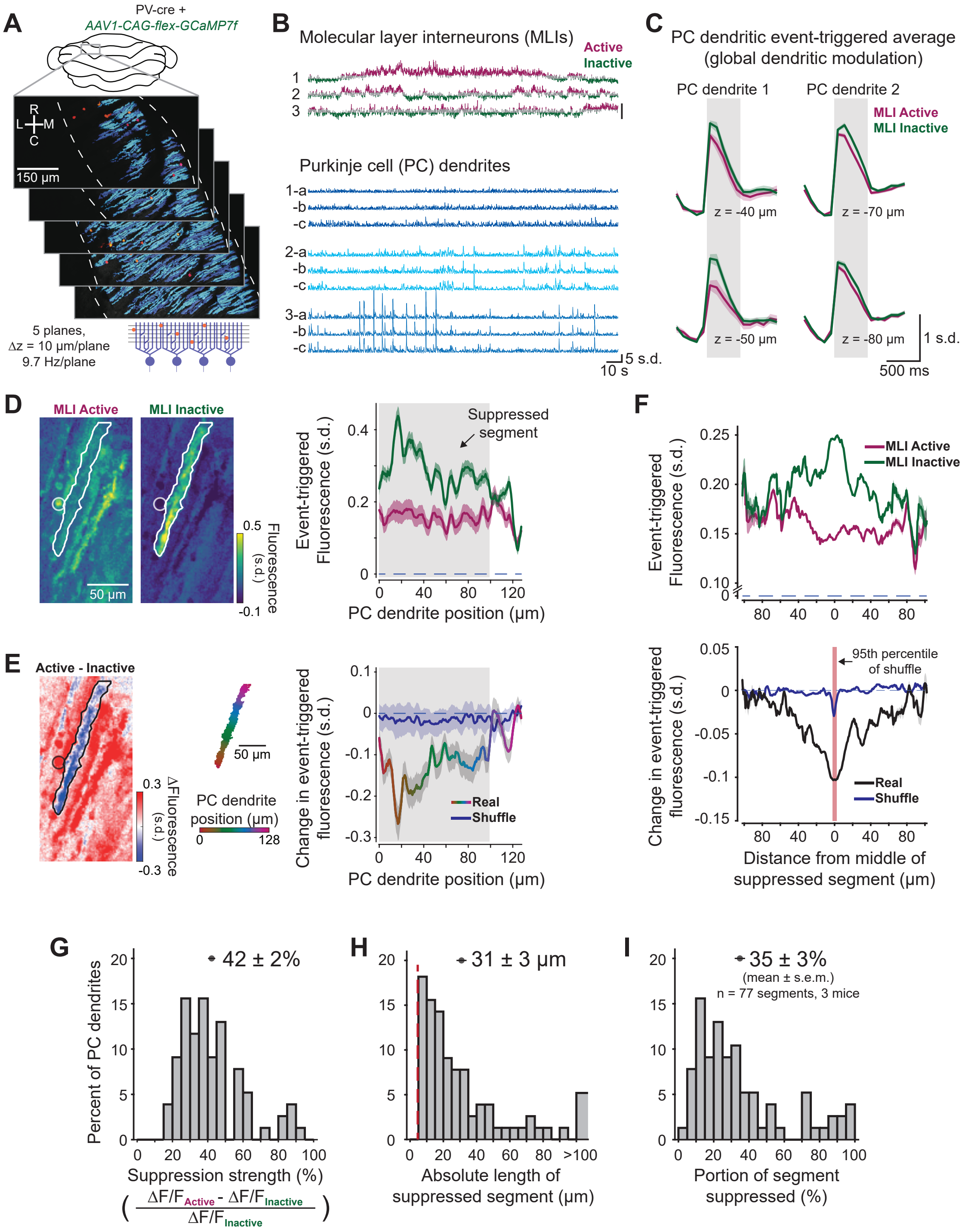

Figure 8: Dendritic gating of Purkinje cells by molecular layer interneurons *in vivo*.

Figure 8: **A**. Multi-plane 2-photon calcium imaging of molecular layer interneurons (red) and Purkinje cell dendrites (blue). **B**. Example traces of MLIs (top, active and inactive states in purple and green) and Purkinje cell dendrites recorded in multiple planes marked a, b, and c (bottom). **C**. Two examples of mean climbing fiber-evoked Ca^{2+} signal, in two planes, in MLI-active (purple) and inactive (green) states. **D**. Spatial event-triggered map of area surrounding Purkinje cell dendrites (contoured region of interest) when a nearby MLI (circle projected from different plane) was in an active state or inactive state (left). Spatial profile of event-triggered fluorescence is shown on the right. **E**. Difference heatmap image (left) and spatial profile (rainbow trace, right) of event-triggered fluorescence of same dendrite shown in panel D. Shuffled trace (blue) was computed on a split of odd and even event indices. **F**. Mean spatial profile of event-triggered fluorescence in MLI-active and MLI-inactive condtions (top) and difference traces (bottom) aligned to middle of suppressed segment (black trace, $n = 77$ regions from $N = 3$ recordings in 3 mice). Shuffled trace (blue) was computed, as above, on an odd-even split. Suppressed regions smaller than the 95th percentile of the shuffle data (red region, 4.6 μm) were excluded. **G**. Histogram of strength of suppression calculated only within modulated segments. **H**. Histogram of suppressed segment lengths (red line shows 95th percentile of the shuffle). **I**. Histogram of suppressed segment lengths expressed as a percent of total segment length. Data are shown as mean ± S.E.M.

Purkinje cell dendritic tree with a spatial extent that is comparable to the length of individual spiny branchlets. This provides strong experimental evidence for a specific biological implementation of one of the principal design features of the DGN, namely local dendritic suppression of principal units by interneurons.

3 Discussion

Identifying the biologically plausible learning rules that mediate the modification of connections in neural networks is a key goal of both experimental and theoretical neuroscience. Here, we describe a new class of learning rules called Dendritic Gated Networks (DGNs). Each unit in each layer of a DGN consists of dendritic branches that are gated on and off by interneurons, and all units in all layers receive the same feedback learning signal. We show that the DGN has key advantages over existing learning algorithms, particularly in terms of learning speed and resilience to forgetting, tested across a range of learning tasks. We also show using *in vivo* experiments that key elements of the DGN architecture may be implemented in biological networks. These results suggest that DGNs may be widely useful in the machine learning community, and also suggest that this learning rule may be implemented in biological networks such as the cerebellum and other neural circuits with a similar network architecture.

3.1 Comparison of DGNs to other learning algorithms

The DGN network architecture differs from traditional learning algorithms (e.g. backprop) as well as the algorithm on which it was based (Gated Linear Networks) in several important ways. Traditional learning algorithms like backprop map input to output in stages, with the input gradually transformed, until eventually, in the output layer, the relevant features are easy to extract. There is certainly some evidence for this hierarchical strategy being implemented in the brain. It is, for example, much harder to extract which face a person is looking at from activity in visual area V1 than in fusiform face area [86, 87]. While this strategy for computing is reasonable, it has a downside: the relationship between activity in intermediate layers and activity in the output layer is highly nontrivial, which makes it especially hard for the brain to determine how weights in intermediate layers should change.

Despite the inherent complexity of this strategy, biologically plausible learning rules that implement it have been proposed [15–27]. The DGN algorithm takes a different approach from any of these. With this architecture, dendritic branches are gated on and off via a random (and fixed) linear transformation of their input. The summed activity of these branches forms the prediction of the neuron, which gets adjusted over time via a delta rule. Consequently, all neurons predict the same target; and each layer improves upon the predictions of the previous layer.

The DGN also differs from and improves over the algorithm on which it was based – the Gated Linear Network (GLN) [29, 30]. In particular, the GLN requires a bank of weights for each neuron, with the input choosing which one the neuron should use – something that seems extremely difficult for the brain to implement. The DGN, however, replaces the library of weights with gated dendritic branches, an innovation essential for biological plausibility. Thus, although the DGN is conceptually related to the GLN, from the point of view of neuroscience it has a critical new component which makes it, unlike the GLN, relevant to the brain.

3.2 Implementation in cerebellar circuits

The architecture and exceptional efficiency of learning exhibited by DGNs suggests that this algorithm may also be implemented in biological networks. Specifically, several key features of the DGN are recapitulated in the functional architecture of the cerebellum. First, the cerebellum receives a clear and global feedback signal in the form of the climbing fiber input to Purkinje cells and cerebellar nuclear neurons that is the principal driver of learning in the cerebellar circuit [88]. Second, the principal neurons of the cerebellar cortex, Purkinje cells, exhibit linear encoding of their inputs due to their high baseline firing rates and unique biophysical properties [36, 89]. Finally, molecular layer interneurons, which are known to target dendritic branches of Purkinje cells, are likely candidates to mediate branch-specific dendritic inhibition [46, 48, 90].

A key prediction of the DGN that would bolster its biological plausibility is that interneurons should gate activity in single dendritic branches of principal cells. Here, we provide the first *in vivo* evidence that molecular layer interneurons can produce inhibition of dendritic calcium signals on the level of single dendritic branches in Purkinje cells, a longstanding, but until now untested, prediction of anatomical [91–94] and theoretical [63, 90, 95] work. By simultaneously imaging dendritic calcium signals in Purkinje cells and activity in neighboring MLIs, we show that MLI activity can substantially decrease dendritic calcium signals in Purkinje cells. Previous *in vitro* work has shown that even modest inhibition of dendritic calcium signals (on order of 20%) can completely abolish cerebellar plasticity [96], suggesting that the suppression we observe is capable of abolishing learning. These MLI-driven effects were not distributed equally across PC dendritic arbors. Indeed, suppression of dendritic calcium signals was often restricted to individual dendritic branches of Purkinje cells, even as neighboring regions of the same dendritic arbor were unaffected. The profound local suppression of these climbing fiber-driven signals suggest that MLI-driven inhibtion is also capable of suppressing parallel fiber-driven input to Pukrinje cells, which are relatively much weaker [97]. Thus, it is likely that MLIs can gate both input and learning to single Purkinje cell dendritic branches. In summary, our demonstration that MLIs can modulate the Purkinje cell dendritic calcium signals on the spatial scale of a single dendritic spiny branchlet strongly supports the DGN gating prediction. We note that the binary on/off gating exhibited by the DGN is challenging to implement biologically; far more likely are softer gates, where the amount of gating is a function of the input. To determine how soft gates affected DGN performance, we performed simulations on the permuted MNIST and inverse kinematics tasks, and the results were virtually identical to the ones with on/off gates (data not shown), emphasizing the flexibility of DGN implementation in the brain. Our experimental data (together with previous work linking cerebellar functional architecture to features of the DGN) provides strong support for the idea that a DGN-like algorithm is implemented in cerebellar circuits.

The architecture of the DGN makes several further predictions about how the DGN may map onto the cerebellum. A key prediction that awaits experimental validation is that the activity of MLIs should depend predominantly on parallel fiber input and the input-output relationship of these interneurons should change very slowly relative to the timescale over which Purkinje cells learn, which can be measured in single trials [98]. Assessing the stability of the parallel fiber-mediated input-output relationship of MLIs is difficult because granule cells are known to exhibit learning-related changes in activation [99, 100]. Thus, further experiments to determine the stability of parallel fiber-MLI and parallel fiber-Purkinje simple spike input-output relationships over learning will need to account for changes in the firing patterns of cerebellar afferents. Another prediction of the DGN is that the parallel fiber connectivity pattern in the VOR or similar tasks should carry information about architecture. If parallel fiber-Purkinje cell connectivity is smooth and diverse (Fig. 7D, top panels), this supports a DGN-like implementation. If, on the other hand, connectivity is smooth and stereotyped (Fig. 7D, bottom panels) or noisy with no other temporal structure (Fig. S4), this argues for alternative implementations, such as an MLP. Distinguishing between these alternatives experimentally would require recording from populations of individual granule cells during VOR learning, then mapping the synaptic strength between those same granule cells and Purkinje cells.

3.3 DGNs in other neural circuits

While DGNs exhibit features that make them particularly well-suited for implementation in the cerebellum, the general principles of this learning rule may be applicable to a variety of brain circuits. In particular, the gating of dendritic signals that we demonstrate in Purkinje cells may also be a feature of cortical networks, given the branch-specific innervation of interneuron axons that has been documented in many cortical circuits [101]. Such generalization would require some modifications to implement the algorithm; for instance, the learning rule will have to change because activity in the cortex is far from linear.

3.4 Conclusions

In summary, Dendritic Gated Networks are strong candidates for biological networks – and not just in the cerebellum; they could be used anywhere there is approximately feedforward structure. They come with two highly desirable features: rapid, data-efficient learning, and biologically plausible learning. Furthermore, they suggest a novel role for inhibitory neurons, which is that they are used for gating dendritic branches. We anticipate that the strong, experimentally testable, predictions may inspire investigations in many brain circuits where rapid learning may invoke a DGN algorithm.

4 Methods

4.1 Classification tasks

The network we use in our model is described in Eqs. (1) and (3), and the learning rules are given in Eq. (4). For regression (VOR and inverse kinematics), we use the identity function for ϕ (Eq. (1)), and the square loss (Eq. (5)), resulting in the update rule given in Eq. (6). For classification (the simple task described in Fig. 3 and permuted MNIST), the network computes probabilities, so ϕ needs to be bounded, and a square loss is not appropriate. Here we provide details for this case.

For classification we use a standard sigmoid function, $\sigma(z) = e^z/(1+e^z)$, albeit modified slightly,

$$\phi(z) = \text{CLIP}_{\epsilon}^{1-\epsilon}\big(\sigma(z)\big) \tag{9}$$

where $\text{CLIP}_a^b(\cdot)$ clips values between a and b (so the right hand side is zero if $\sigma(z)$ is smaller than ϵ or larger than $1-\epsilon$). Clipping is used for bounding the loss as well as the gradients, which helps with numerical stability. It also enables a worst-case regret analysis [29, 30]. We set ϵ to 0.01, so neural activity lies between 0.01 and 0.99.

The square loss is not appropriate in this case, so instead we use the binary cross-entropy loss: the loss of neuron i in layer k is given by

$$\ell(r^*, r_{k,i}) = -r^* \log r_{k,i} - (1-r^*) \log\big(1 - r_{k,i}\big)\,. \tag{10}$$

Consequently, the update rule for the weights, Eq. (4), is (after a small amount of algebra)

$$\Delta w^b_{k,ij} = \eta g^b_{k,i}(\mathbf{x}) \mathbb{1}(\epsilon < r_{k,i} < 1-\epsilon) \left(r^* - r_{k,i}\right) h_{k-1,j} \tag{11}$$

where $\mathbb{1}(\cdot)$ is 1 when its argument is true and 0 otherwise. The fact that learning is zero when $r_{k,i}$ is outside the range $[\epsilon, 1-\epsilon]$ follows because $d\phi(z)/dz = 0$ when z is outside this range (see Eq. (9)). This ensures that learning saturates when weights become too large (either positive or negative). However, this can cause problems if the output is very wrong; that is, when $r^* = 1$ and $r_{k,i} < \epsilon$ or $r^* = 0$ and $r_{k,i} > 1-\epsilon$. To address this, we allow learning in this regime. We can do that compactly by changing the learning rule to

$$\Delta w^b_{k,ij} = \eta g^b_{k,i}(\mathbf{x}) \mathbb{1}(|r^* - r_{k,i}| > \epsilon) \left(r^* - r_{k,i}\right) h_{k-1,j} \, . \tag{12}$$

Essentially, this rule says: stop learning when $r_{k,i}$ is within ϵ of r^*.

For a compact summary of the equations (given as pseudocode), see Supplementary Algorithms S1 and S2.

4.2 Simulations

Here we provide details of our simulations. Simulations were written using JAX [102], the DeepMind JAX Ecosystem [103], and Colab [104].

Permuted MNIST. We adopt the pixel-permuted MNIST benchmark [57, 60], which is a sequence of MNIST digit classification tasks with different pixel permutations. Each task consists of 60,000 training images and 10,000 test images; all images are deskewed. Models are trained sequentially across 10 tasks, performing a single pass over all 60,000 training examples for each of the tasks. We provide the implementation details below; the parameters swept during a grid search are given in Supplementary Table S2.

DGN. We use networks composed of 100 and 20 units in the hidden layers and a single linear neuron for the output. All neurons in the hidden layers have 10 dendritic branches. The output of the network is determined by the last neuron. MNIST has 10 classes, each corresponding to a digit. Therefore, we use 10 DGN networks, each encoding the probability of a distinct class. These networks are updated during training using a learning rate $\eta = 10^{-2}$. During testing, the class with the maximum probability is chosen. Images are scaled and shifted so that the input range is $[-1, 1]$. The gating vectors, $\mathbf{v}^b_{k,i}$, are chosen randomly on the unit sphere, which can be achieved by sampling from an isotropic Normal distribution and then dividing by the L2 norm. The biases, $\theta^b_{k,i}$, are drawn independently from a zero mean Gaussian with standard deviation 0.05.

MLP and EWC. We use a ReLu network with 1000 and 200 neurons in the hidden layers and 10 linear output units with cross entropy loss. In this setting, the MLP and

EWC have the same number of neurons as the DGN, but fewer connections. We use the ADAM optimization method [105] with a learning rate $\eta = 10^{-4}$ (see Supplementary Table S2 for details of the hyperparameter optimization), in conjunction with dropout. We use mini-batches of 20 data points. For EWC, we draw 100 samples for computing the Fisher matrix diagonals and set the regularization constant to 10^3.

Inverse Kinematics. Each DGN network has 20 Purkinje cells and one linear, non-gated, output neuron, and we vary the number of branches. We use a quadratic loss, as in Eq. (5), a learning rate $\eta = 10^{-5}$, and we run for 2000 epochs (2000 passes over the dataset). The inputs are centered at 0 and scaled to unit variance per dimension, and the targets are scaled so that they lie between 0 and 1. The reported MSEs are computed on the test set based on inverse transformed predictions (thus undoing the target scaling). The gating parameters are chosen in the same way as for the MNIST simulations described above.

We discovered that the training set of the SARCOS dataset (downloaded from `http://www.gaussianprocess.org/gpml/data/` on 15 December 2020) includes test instances. To the best of our knowledge, other recent studies using the SARCOS dataset [106, 107] reported results with this

train/test setting. This means that the reported errors are measures of capacity rather than generalization. We compare the performance of DGN against the best known SARCOS results in Supplementary Table S1 using the existing train/test split.

VOR. The gating parameters $v^b_{k,ij}$ and $\theta^b_{k,i}$ (Eq. (3)), were drawn independently from the standard normal distribution. The learning rates were $\eta = 10^{-5}$ for the DGN and $\eta = 0.03$ for the MLP.

4.3 Animal experiments

Animal housing and surgery All animal procedures were approved by the local Animal Welfare and Ethical Review Board and performed under license from the UK Home Office in accordance with the Animals (Scientific Procedures) Act 1986 and generally followed procedures described previously [108]. Briefly, we used PV-Cre mice (B6;129P2-Pvalbtm1(cre)Arbr/J) [109] maintained on a C57/BL6 background. Mice were group housed before and after surgery and maintained on a 12:12 day-night cycle. Surgical procedures were similar to those described in [108], except that we injected Cre-dependent GCaMP7f (pGP-AAV-CAG-FLEX-jGCaMP7f-WPRE [serotype 1]; [110]) diluted from its stock titer to a final concentration of 3 x 10^{11} GC/ml (∼1:25). After mice had recovered from surgery, they were acclimated to the recording setup and expression-checked before beginning recordings.

Two-photon calcium imaging data acquisition and processing Imaging experiments were performed using a 16x/0.8 NA objective (Nikon) mounted on a Sutter MOM microscope equipped with the Resonant Scan Box module. A Ti:Sapphire laser tuned to 930 nm (Mai Tai, Spectra Physics) was raster scanned using a resonant scanning galvanometer (8 kHz, Cambridge Technologies) and images were collected at

512x256 pixel resolution over fields of view of 670x335 μm per plane at an average power of 30-70 mW. Volumetric imaging across 5 planes spaced by 10 μm (40 μm depth range per recording, total depth ranging 10-70 μm below pial surface) were performed using a P-726 PIFOC High-Load Objective Scanner (Physik Instruments) at an effective volume rate of 9.7 Hz. The microscope was controlled using ScanImage (Version 5.6, Vidrio Technologies) and tilted to 12.5 degrees such that the objective was orthogonal to the surface of the brain and coverglass.

Regions of interest (ROIs) corresponding to single MLI somata and Purkinje cell dendrites, which were easily distinguishable based on their shape, were identified using Suite2p software [111] for initial source extraction and custom-written software for subsequent analyses. Fluorescence time series were computed as $(F - F0)/F0$ where F was the signal measured at each point in time and $F0$ is the 8th percentile of a rolling average baseline surrounding each data point (2000 frames for MLIs and 10 frames for Purkinje cell dendrites). A neuropil correction coefficient of 0.5 (50 percent of neuropil signal output from Suite2p) was applied to MLI ROIs. A range of baseline durations and neuropil correction coefficients were tested and varying these parameters did not alter the main findings. Following these calculations, fluorescence signals were z-scored to facilitate comparisons across neurons. Signals from the same Purkinje cell dendrites recorded in multiple planes were identified based on a correlation threshold (>0.5, followed by manual confirmation) and analyzed independently for dendritic modulation experiments. Event times in dendrites were detected using MLspike [112].

MLI gating of Purkinje cell dendritic signals For dendritic modulation experiments, active and inactive MLI states were defined as imaging frames where activity in an MLI deviated more than 0.5 standard deviations above or below the mean, respectively (Fig. S5). Using a higher threshold yielded similar results but resulted in fewer dendritic events in each condition. After identifying these states, we compared the magnitudes of isolated dendritic events in these two conditions for Purkinje cells within a 300×100 μm ellipse centered on each MLI whose major axis is parallel to that of Purkinje cell dendrites in the field of view (approximately rostrocaudal). Ellipse dimensions were chosen to approximate the known rostro-caudal and mediolateral spread of MLI axons [84, 85]. Only isolated events in Purkinje cell dendrites (defined as those that occurred more than 500 ms before and after any other events) were analyzed, and fluorescence event magnitudes were calculated over the 5 frames (∼500 ms) after each event for initial identification of MLI-modulated dendrites. Because analysis of each recording involved many thousands of comparisons, we assessed significance differences between Purkinje cell dendrite event magnitudes in MLI active and inactive states with a significance threshold of 0.05 that was corrected for multiple comparisons using false discovery rate threshold of 5% [113].

Motion-corrected fluorescence movies that were used for pixel-wise analysis of dendritic subregions were pre-processed by first correcting for slow fluctuations in fluorescence, which was done by computing $(F - F_0)/F_0$ where F was the signal measured at each point in time and F_0 is the 8 percentile of a rolling average baseline surrounding each data point (2000 frames), and then z-scoring. Purkinje cell dendritic ROIs defined in Suite2p were segmented into 1 μm increments by fitting a 4th degree polynomial to each ROI, grouping ROI pixels closest to regular spaced points along this fit line, computing a weighted average of these pixels based on the pixel weights assigned by Suite2p, and smoothing over 5 μm. Activity profiles in MLI active and inactive states were subtracted and used to generate spatial suppression profiles for each Purkinje cell dendrite and error bars were generated from the summed variances in active and inactive conditions. Shuffled distributions were generated by replacing active and inactive conditions with an odd-even event split of each of these groups, yielding two distributions comprising of 1/2 MLI-active and 1/2 MLI-inactive events for each dendritic segment. Significantly modulated dendritic segments were defined as the longest region for each dendritic segment in which the 95% confidence interval of the difference trace was less than zero. To account for false positive dendritic segments that would be identified by finding minima in noise, we performed this identification procedure on our shuffled data and excluded identified dendritic segments in our real data that were smaller than the 95th percentile of these fictive segments (4.6 μm).

Code availability. We provide pseudo code in Supplementary Algorithms S1 andS2. A simple python implementation can be accessed via `https://github.com/deepmind/deepmind-research/blob/master/gated_linear_networks/colabs/dendritic_gated_network.ipynb`.

Data availability. The data that support the findings of this study are available from the corresponding authors upon reasonable request. Additional analysis made use of standard publicly available benchmarks including MNIST [114] and SARCOS (`http://www.gaussianprocess.org/gpml/data/`).

Acknowledgements. We thank Timothy Lillicrap, Gregory Wayne, Eszter Vértes, and Brendan Bicknell for valuable feedback. Michael Häusser is supported by the Wellcome Trust and the European Research Council. Peter Latham is supported by the Gatsby Charitable Foundation and the Wellcome Trust.

Author contributions. ES and AGB developed the computational model with advice from JV, CC, PEL, DB, and MHut. AGB, ES, and SK performed simulation experiments and analysis with advice from CC and PEL. DK and MBea acquired and analyzed neuronal data with advice from MHäu. PEL, AGB, ES, and DK wrote the paper with help from all other authors. AGB and ES managed the project with support from MBot, CC, JV, and DB.

Competing Interests. The authors declare no competing interests.

References

[1] Burke, C. J. *et al.* Layered reward signalling through octopamine and dopamine in drosophila. *Nature* **492**, 433–437 (2012).

[2] Liu, C. *et al.* A subset of dopamine neurons signals reward for odour memory in drosophila. *Nature* **488**, 512–516 (2012).

[3] Schmidhuber, J. Deep learning in neural networks: An overview. *Neural networks* **61**, 85–117 (2015).

[4] Grossberg, S. Competitive learning: From interactive activation to adaptive resonance. *Cogn. Sci.* **11**, 23–63 (1987).

[5] Crick, F. The recent excitement about neural networks. *Nature* **337**, 129–132 (1989).

[6] Roelfsema, P. R. & Holtmaat, A. Control of synaptic plasticity in deep cortical networks. *Nature Reviews Neuroscience* **19**, 166 (2018).

[7] Whittington, J. C. & Bogacz, R. Theories of error back-propagation in the brain. *Trends in cognitive sciences* **23**, 235–250 (2019).

[8] Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J. & Hinton, G. Backpropagation and the brain. *Nature Reviews Neuroscience* **21**, 335–346 (2020).

[9] Kelley, H. J. Gradient theory of optimal flight paths. *Ars Journal* **30**, 947–954 (1960).

[10] Linnainmaa, S. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki* 6–7 (1970).

[11] Rumelhart, D. E., , G. E. & Williams, R. J. Learning representations by back-propagating errors. In Anderson, J. A. & Rosenfeld, E. (eds.) *Neurocomputing: Foundations of Research*, 696–699 (MIT Press, Cambridge, MA, USA, 1988).

[12] French, R. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences* **3**, 128–135 (1999).

[13] Herzfeld, D. J., Vaswani, P. A., Marko, M. K. & Shadmehr, R. A memory of errors in sensorimotor learning. *Science* **345**, 1349–1353 (2014).

[14] Wolpert, D. M. & Flanagan, J. R. Computations underlying sensorimotor learning. *Current opinion in neurobiology* **37**, 7–11 (2016).

[15] Lillicrap, T. P., Cownden, D., Tweed, D. B. & Akerman, C. J. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications* **7**, 13276 (2016). URL `http://www.nature.com/articles/ncomms13276`.

[16] Akrout, M., Wilson, C., Humphreys, P., Lillicrap, T. & Tweed, D. B. Deep learning without weight transport. In *Advances in Neural Information Processing Systems*, vol. 6, e22901 (2017).

[17] Guerguiev, J., Lillicrap, T. P. & Richards, B. A. Towards deep learning with segregated dendrites. *eLife* **6**, e22901 (2019).

[18] Sacramento, J., Costa, R. P., Bengio, Y. & Senn, W. Dendritic cortical microcircuits approximate the backpropagation algorithm. In *Advances in Neural Information Processing Systems*, 8721–8732 (2018).

[19] Payeur, A., Guerguiev, J., Zenke, F., Richards, B. A. & Naud, R. Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. *bioRxiv* (2020). URL `https://www.biorxiv.org/content/early/2020/03/31/2020.03.30.015511`.

[20] Samadi, A., Lillicrap, T. P. & Tweed, D. B. Deep learning with dynamic spiking neurons and fixed feedback weights. *Neural Computation* **29**, 578–602 (2017).

[21] Belilovsky, E., Eickenberg, M. & Oyallon, E. Greedy layerwise learning can scale to ImageNet. In *Proceedings of the 36th International Conference on Machine Learning*, vol. 97, 583–593 (PMLR, 2019). URL `http://proceedings.mlr.press/v97/belilovsky19a.html`.

[22] Nøkland, A. Direct feedback alignment provides learning in deep neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, 1045–1053 (Curran Associates Inc., Red Hook, NY, USA, 2016).

[23] Nøkland, A. & Eidnes, L. H. Training Neural Networks with Local Error Signals. In *International Conference on Machine Learning*, 4839–4850 (PMLR, 2019). URL `http://proceedings.mlr.press/v97/nokland19a.html`. ISSN: 2640-3498.

[24] Löwe, S., O'Connor, P. & Veeling, B. Putting an end to end-to-end: Gradient-isolated learning of representations. In *Advances in Neural Information Processing Systems*, 3033–3045 (2019).

[25] Pogodin, R. & Latham, P. E. Kernelized information bottleneck leads to biologically plausible 3-factor hebbian learning in deep networks. *arXiv preprint arXiv:2006.07123* (2020). `2006.07123`.

[26] Podlaski, W. F. & Machens, C. K. Biological credit assignment through dynamic inversion of feedforward networks (2020). `2007.05112`.

[27] Golkar, S., Lipshutz, D., Bahroun, Y., Sengupta, A. M. & Chklovskii, D. B. A biologically plausible neural network for local supervision in cortical microcircuits (2020). `2011.15031`.

[28] Clark, D. G., Abbott, L. & Chung, S. Credit assignment through broadcasting a global error vector. *arXiv preprint arXiv:2106.04089* (2021).

[29] Veness, J. *et al.* Online learning with gated linear networks. *arXiv preprint arXiv:1712.01897* (2017).

[30] Veness, J. *et al.* Gated linear networks. *Proceedings of the AAAI Conference on Artificial Intelligence (To Appear)* (2021).

[31] Budden, D. *et al.* Gaussian gated linear networks. In *Advances in Neural Information Processing Systems* (2020).

[32] Sezener, E., Hutter, M., Budden, D., Wang, J. & Veness, J. Online learning in contextual bandits using gated linear networks. In *Advances in Neural Information Processing Systems* (2020).

[33] Wang, J., Sezener, E., Budden, D., Mutter, M. & Veness, J. A combinatorial perspective on transfer learning. In *Advances in Neural Information Processing Systems* (2020).

[34] Widrow, B. & Hoff, M. E. Adaptive switching circuits. In *1960 IRE WESCON Convention Record, Part 4*, 96–104 (IRE, New York, 1960).

[35] Tsuda, B., Tye, K. M., Siegelmann, H. T. & Sejnowski, T. A modeling framework for adaptive lifelong learning with transfer and savings through gating in the prefrontal cortex. *Proceedings of the National Academy of Sciences* **117**, 29872–29882 (2020).

[36] Llinás, R. & Sugimori, M. Electrophysiological properties of in vitro purkinje cell somata in mammalian cerebellar slices. *The Journal of physiology* **305**, 171–195 (1980).

[37] Walter, J. T. & Khodakhah, K. The linear computational algorithm of cerebellar purkinje cells. *Journal of Neuroscience* **26**, 12861–12872 (2006).

[38] Chen, S., Augustine, G. J. & Chadderton, P. Serial processing of kinematic signals by cerebellar circuitry during voluntary whisking. *Nature communications* **8**, 1–13 (2017).

[39] Eccles, J. Functional meaning of the patterns of synaptic connections in the cerebellum. *Perspectives in biology and medicine* **8**, 289–310 (1965).

[40] Dizon, M. J. & Khodakhah, K. The role of interneurons in shaping purkinje cell responses in the cerebellar cortex. *Journal of Neuroscience* **31**, 10463–10473 (2011).

[41] Brown, A. M. *et al.* Molecular layer interneurons shape the spike activity of cerebellar purkinje cells. *Scientific reports* **9**, 1–19 (2019).

[42] Andersen, P., Eccles, J. & Voorhoeve, P. Postsynaptic inhibition of cerebellar purkinje cells. *Journal of neurophysiology* **27**, 1138–1153 (1964).

[43] Mittmann, W., Koch, U. & Häusser, M. Feed-forward inhibition shapes the spike output of cerebellar purkinje cells. *The Journal of physiology* **563**, 369–378 (2005).

[44] Häusser, M. & Clark, B. A. Tonic synaptic inhibition modulates neuronal output pattern and spatiotemporal synaptic integration. *Neuron* **19**, 665–678 (1997).

[45] Oldfield, C. S., Marty, A. & Stell, B. M. Interneurons of the cerebellar cortex toggle purkinje cells between up and down states. *Proceedings of the National Academy of Sciences* **107**, 13153–13158 (2010).

[46] Callaway, J. C., Lasser-Ross, N. & Ross, W. N. Ipsps strongly inhibit climbing fiber-activated [ca2+] i increases in the dendrites of cerebellar purkinje neurons. *Journal of Neuroscience* **15**, 2777–2787 (1995).

[47] Wulff, P. *et al.* Synaptic inhibition of purkinje cells mediates consolidation of vestibulo-cerebellar motor learning. *Nature neuroscience* **12**, 1042–1049 (2009).

[48] Gaffield, M. A., Rowan, M. J., Amat, S. B., Hirai, H. & Christie, J. M. Inhibition gates supralinear ca2+ signaling in purkinje cell dendrites during practiced movements. *Elife* **7**, e36246 (2018).

[49] Carpenter, G. A. & Grossberg, S. The art of adaptive pattern recognition by a self-organizing neural network. *Computer* **21**, 77–88 (1988).

[50] McCloskey, M. & Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation* **24**, 109 – 165 (1989). URL `http://www.sciencedirect.com/science/article/pii/S0079742108605368`.

[51] Robins, A. V. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connect. Sci.* **7**, 123–146 (1995).

[52] Caruana, R. Multitask learning. *Machine Learning* **28**, 41–75 (1997). URL `https://doi.org/10.1023/A:1007379606734`.

[53] Rebuffi, S.-A., Kolesnikov, A., Sperl, G. & Lampert, C. H. icarl: Incremental classifier and representation learning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). URL `http://dx.doi.org/10.1109/CVPR.2017.587`.

[54] Donahue, J. *et al.* Decaf: A deep convolutional activation feature for generic visual recognition. *CoRR* **abs/1310.1531** (2013). URL `http://arxiv.org/abs/1310.1531`. `1310.1531`.

[55] Razavian, A. S., Azizpour, H., Sullivan, J. & Carlsson, S. Cnn features off-the-shelf: An astounding baseline for recognition. *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops* (2014). URL `http://dx.doi.org/10.1109/CVPRW.2014.131`.

[56] Girshick, R., Donahue, J., Darrell, T. & Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition* (2014). URL `http://dx.doi.org/10.1109/CVPR.2014.81`.

[57] Kirkpatrick, J. *et al.* Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences* **114**, 3521–3526 (2017). URL `https://www.pnas.org/content/114/13/3521`. `https://www.pnas.org/content/114/13/3521.full.pdf`.

[58] Zenke, F., Poole, B. & Ganguli, S. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, 3987–3995 (JMLR.org, 2017).

[59] Schwarz, J. *et al.* Progress &; compress: A scalable framework for continual learning. In Dy, J. & Krause, A. (eds.) *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, 4528–4537 (PMLR, Stockholmsmässan, Stockholm Sweden, 2018). URL `http://proceedings.mlr.press/v80/schwarz18a.html`.

[60] Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A. & Bengio, Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks (2013). `1312.6211`.

[61] Kitamura, K. & Häusser, M. Dendritic calcium signaling triggered by spontaneous and sensory-evoked climbing fiber input to cerebellar purkinje cells in vivo. *Journal of Neuroscience* **31**, 10847–10858 (2011).

[62] Marr, D. A theory of cerebellar cortex. *The Journal of Physiology* **202**, 437–470.1 (1969). URL `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1351491/`.

[63] Albus, J. S. A theory of cerebellar function. *Mathematical Biosciences* **10**, 25–61 (1971). URL `http://www.sciencedirect.com/science/article/pii/0025556471900514`.

[64] Ito, M. & Kano, M. Long-lasting depression of parallel fiber-purkinje cell transmission induced by conjunctive stimulation of parallel fibers and climbing fibers in the cerebellar cortex. *Neuroscience Letters* **33**, 253 – 258 (1982). URL `http://www.sciencedirect.com/science/article/pii/0304394082903809`.

[65] Narain, D., Remington, E. D., Zeeuw, C. I. D. & Jazayeri, M. A cerebellar mechanism for learning prior distributions of time intervals. *Nature Communications* **9**, 469 (2018).

[66] Shadmehr, R. Population coding in the cerebellum and its implications for learning from error. *bioRxiv* (2020).

[67] Pedroarena, C. M. & Schwarz, C. Efficacy and short-term plasticity at gabaergic synapses between purkinje and cerebellar nuclei neurons. *Journal of Neurophysiology* **89**, 704–715 (2003).

[68] Pedroarena, C. M. Short-term plasticity at purkinje to deep cerebellar nuclear neuron synapses supports a slow gain-control mechanism enabling scaled linear encoding over second-long time windows. *bioRxiv* 749259 (2019).

[69] Morishita, W. & Sastry, B. Postsynaptic mechanisms underlying long-term depression of gabaergic transmission in neurons of the deep cerebellar nuclei. *Journal of Neurophysiology* **76**, 59–68 (1996).

[70] Aizenman, C. D., Manis, P. B. & Linden, D. J. Polarity of long-term synaptic gain change is related to postsynaptic spike firing at a cerebellar inhibitory synapse. *Neuron* **21**, 827–835 (1998).

[71] Aizenman, C. D. & Linden, D. J. Rapid, synaptically driven increases in the intrinsic excitability of cerebellar deep nuclear neurons. *Nature neuroscience* **3**, 109–111 (2000).

[72] Zheng, N. & Raman, I. M. Synaptic inhibition, excitation, and plasticity in neurons of the cerebellar nuclei. *The Cerebellum* **9**, 56–66 (2010).

[73] Hull, C. Prediction signals in the cerebellum: beyond supervised motor learning. *Elife* **9**, e54073 (2020).

[74] Kawato, M., Furukawa, K. & Suzuki, R. A hierarchical neural-network model for control and learning of voluntary movement. *Biological cybernetics* **57**, 169–185 (1987).

[75] Wolpert, D. M., Miall, R. C. & Kawato, M. Internal models in the cerebellum. *Trends in cognitive sciences* **2**, 338–347 (1998).

[76] Vijayakumar, S. & Schaal, S. Locally weighted projection regression: An o (n) algorithm for incremental real time learning in high dimensional space. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, vol. 1, 288–293 (2000).

[77] Robinson, D. Adaptive gain control of vestibuloocular reflex by the cerebellum. *Journal of neurophysiology* **39**, 954–969 (1976).

[78] Miles, F. & Lisberger, S. Plasticity in the vestibulo-ocular reflex: a new hypothesis. *Annual review of neuroscience* **4**, 273–299 (1981).

[79] Lac, S., Raymond, J. L., Sejnowski, T. J. & Lisberger, S. G. Learning and memory in the vestibulo-ocular reflex. *Annual review of neuroscience* **18**, 409–441 (1995).

[80] Ito, M. Cerebellar learning in the vestibulo-ocular reflex. *Trends in cognitive sciences* **2**, 313–321 (1998).

[81] Boyden, E. S., Katoh, A. & Raymond, J. L. Cerebellum-dependent learning: the role of multiple plasticity mechanisms. *Annual review of neuroscience* **27** (2004).

[82] Clopath, C., Badura, A., De Zeeuw, C. I. & Brunel, N. A cerebellar learning model of vestibulo-ocular reflex adaptation in wild-type and mutant mice. *Journal of Neuroscience* **34**, 7203–7215 (2014).

[83] Armstrong, D. & Rawson, J. Activity patterns of cerebellar cortical neurones and climbing fibre afferents in the awake cat. *The Journal of Physiology* **289**, 425–448 (1979).

[84] Rieubland, S., Roth, A. & Häusser, M. Structured connectivity in cerebellar inhibitory networks. *Neuron* **81**, 913–929 (2014).

[85] Wang, W. X. & Lefebvre, J. L. Morphological pseudotime ordering and fate mapping reveals diversification of cerebellar inhibitory interneurons. *bioRxiv* (2020).

[86] Tsao, D. Y., Freiwald, W. A., Knutsen, T. A., Mandeville, J. B. & Tootell, R. B. Faces and objects in macaque cerebral cortex. *Nature neuroscience* **6**, 989–995 (2003).

[87] Zhen, Z., Fang, H. & Liu, J. The hierarchical brain network for face recognition. *PloS one* **8**, e59886 (2013).

[88] Raymond, J. L. & Medina, J. F. Computational Principles of Supervised Learning in the Cerebellum. *Annual Review of Neuroscience* **41**, 233–253 (2018).

[89] Walter, J. T. & Khodakhah, K. The Linear Computational Algorithm of Cerebellar Purkinje Cells. *Journal of Neuroscience* **26**, 12861–12872 (2006). URL `https://www.jneurosci.org/content/26/50/12861`.

[90] Jörntell, H., Bengtsson, F., Schonewille, M. & De Zeeuw, C. I. Cerebellar molecular layer interneurons–computational properties and roles in learning. *Trends in neurosciences* **33**, 524–532 (2010).

[91] Sotelo, C. Molecular layer interneurons of the cerebellum: developmental and morphological aspects. *The Cerebellum* **14**, 534–556 (2015).

[92] Kim, J. & Augustine, G. J. Molecular layer interneurons: key elements of cerebellar network computation and behavior. *Neuroscience* **462**, 22–35 (2021).

[93] Palay, S. L. & Chan-Palay, V. *Cerebellar cortex: cytology and organization* (Springer Science & Business Media, 2012).

[94] Pouzat, C. & Hestrin, S. Developmental regulation of basket/stellate cell→ purkinje cell synapses in the cerebellum. *Journal of Neuroscience* **17**, 9104–9112 (1997).

[95] Dean, P., Porrill, J., Ekerot, C.-F. & Jörntell, H. The cerebellar microcircuit as an adaptive filter: experimental and computational evidence. *Nature Reviews Neuroscience* **11**, 30–43 (2010).

[96] Rowan, M. J. *et al.* Graded control of climbing-fiber-mediated plasticity and learning by inhibition in the cerebellum. *Neuron* **99**, 999–1015 (2018).

[97] Isope, P. & Barbour, B. Properties of unitary granule cell→ purkinje cell synapses in adult rat cerebellar slices. *Journal of Neuroscience* **22**, 9668–9678 (2002).

[98] Yang, Y. & Lisberger, S. G. Purkinje-cell plasticity and cerebellar motor learning are graded by complex-spike duration. *Nature* **510**, 529–532 (2014).

[99] Wagner, M., Kim, T., Savall, J., Schnitzer, M. & Luo, L. Cerebellar granule cells encode the expectation of reward. *Nature* **544** (2017).

[100] Giovannucci, A. *et al.* Cerebellar granule cells acquire a widespread predictive feedback signal during motor learning. *Journal of Neurophysiology* **20**, 727–734 (2017). URL `https://www.nature.com/articles/nn.4531`.

[101] Tremblay, R., Lee, S. & Rudy, B. Gabaergic interneurons in the neocortex: from cellular properties to circuits. *Neuron* **91**, 260–292 (2016).

[102] Bradbury, J. *et al.* JAX: composable transformations of Python+NumPy programs (2018). URL `http://github.com/google/jax`.

[103] Babuschkin, I. *et al.* The DeepMind JAX Ecosystem (2020). URL `http://github.com/deepmind`.

[104] Ekaba, B. Google colaboratory. In: Building Machine Learning and Deep Learning Models on Google Cloud Platform. Apress, Berkeley, CA. (2019). URL `https://doi.org/10.1007/978-1-4842-4470-8_7`.

[105] Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[106] Tanno, R., Arulkumaran, K., Alexander, D., Criminisi, A. & Nori, A. Adaptive neural trees. In Chaudhuri, K. & Salakhutdinov, R. (eds.) *Proceedings of the 36th International Conference on Machine Learning*, vol. 97 of *Proceedings of Machine Learning Research*, 6166–6175 (PMLR, Long Beach, California, USA, 2019). URL `http://proceedings.mlr.press/v97/tanno19a.html`.

[107] Arik, S. O. & Pfister, T. Tabnet: Attentive interpretable tabular learning. *Proceedings of the AAAI Conference on Artificial Intelligence (To Appear)* (2021).

[108] Kostadinov, D., Beau, M., Pozo, M. B. & Häusser, M. Predictive and reactive reward signals conveyed by climbing fiber inputs to cerebellar purkinje cells. *Nature Neuroscience* **22**, 950–962 (2019).

[109] Hippenmeyer, S. *et al.* A developmental switch in the response of drg neurons to ets transcription factor signaling. *PLoS biol* **3**, e159 (2005).

[110] Dana, H. *et al.* High-performance calcium sensors for imaging activity in neuronal populations and microcompartments. *Nature Methods* **16**, 649–657 (2019).

[111] Pachitariu, M. *et al.* Suite2p: beyond 10,000 neurons with standard two-photon microscopy. *Biorxiv* (2017).

[112] Deneux, T. *et al.* Accurate spike estimation from noisy calcium signals for ultrafast three-dimensional imaging of large neuronal populations in vivo. *Nature communications* **7**, 1–17 (2016).

[113] Storey, J. D. A direct approach to false discovery rates. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **64**, 479–498 (2002).

[114] LeCun, Y., Cortes, C. & Burges, C. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* **2** (2010).

Supplementary Information

Difference between GLNs and DGNs

The main difference between the DGNs and Gated Linear Networks (GLNs) is in the interplay of the gating functions and synaptic weights. To understand this, it is helpful to write Eq. (1) in the form

$$r_{k,i} = \phi\left(\sum_{j=0}^{n_{k-1}} \left[\sum_{b=1}^{B_{k,i}} g_{k,i}^b(\mathbf{x}) w_{k,ij}^b\right] h_{k-1,j}\right) . \tag{13}$$

The term in brackets is the effective weight, which, for each j, consists of a sum over branches. The gates, $g_{k,i}^b$, can be either zero or 1; since there are $B_{k,i}$ of them, there are $2^{B_{k,i}}$ possible effective weights. For the GLN, on the other hand, $r_{k,i}$ is given by

$$r_{k,i} = \phi\left(\sum_{j=0}^{n_{k-1}} w_{k,ij}^{\beta(\mathbf{x})} h_{k-1,j}\right) . \tag{14}$$

The difference between this and Eq. (13) is the the term in brackets has been replaced by a single weight, $w_{k,ij}^{\beta(\mathbf{x})}$. However, the index $\beta(\mathbf{x})$ can take on $2^{B_{k,i}}$ values, so there are just as many weights in the GLN as effective weights in the DGN. The value of $\beta(\mathbf{x})$ is determined by the input, and is given by the binary string (suppressing the $\mathbf{x}$ dependence for clarity)

$$\beta = g_{k,i}^1 \, g_{k,i}^2 \dots g_{k,i}^{B_{k,i}} . \tag{15}$$

(So if there were 5 gates, and the 3rd and 5th were on while the others were off, the term on the right would be 00101, and β would be 5.)

In our experience, for similar B, DGNs and GLNs perform equally well. Computationally, DGNs are more memory efficient, as B weight vectors need to be stored per neuron as opposed to 2^B for GLNs. However, this comes at the cost of more operations, as there is an additional sum over branches (the term in brackets in Eq. (13)).

The difference in the number of parameters translates to a difference in inductive bias. GLNs are less prone to catastrophic forgetting compared to DGNs, as only one weight vector per neuron is updated for each input. This, however, means that DGNs are better than GLNs at learning new tasks – so long as there is some shared structure.

Convexity

Here we show that the loss is convex with respect to the weights in the previous layer. Temporarily dropping indices for clarity, the loss, $\ell(r^*, r)$, is given in terms of the weight vector, $\mathbf{w}$, as $\ell(r^*, r) = \ell(r^*, \phi(h))$ with $h = \mathbf{c} \cdot \mathbf{w}$ (see Eq. (1)). If $\ell(r^*, \phi(h))$ is convex in h, then ℓ is also convex in $\mathbf{w}$, since h is a linear function of $\mathbf{w}$.

For quadratic loss, Eq. (5), ϕ is the identity, so $\ell(r^*, \phi(h))) = \frac{1}{2}(r^* - h)^2$. This is obviously convex in h, and so convex in $\mathbf{w}$. For binary cross-entropy loss, Eq. (10), $\phi(h)$ is given by a clipped sigmoid, Eq. (9). When clipped, $\phi(h) = 0$, which is convex. When not clipped, $\phi(h) = \sigma(h) = 1/(1 + e^{-h})$, for which it is easy to show that $\partial^2 \ell(r^*, \phi(h)))/\partial h^2 = \sigma(h)(1 - \sigma(h)) > 0$. Thus, again ℓ is convex in h, and so also in $\mathbf{w}$.

Optimal number of branches

The number of dendritic branches is one of the main factors determining the model capacity of DGNs. Having too few branches results in underfitting, as the network is not flexible enough to learn the underlying function. Having too many branches, on the other hand, can result in memorization, and thus overfitting. We have generally used 10 branches per neuron except in the Inverse Kinematics experiments, where we used up to 5000 branches, as the task measures memorization not generalization.

In Figure S1, we show the average accuracy in the permuted MNIST task as a function of the number of branches. This inverted U-shaped relationship can be observed in most tasks (data not shown). In Figure S2, we show how the MSE improves with an increased number of branches. Because the training and test data is mixed, overfitting is not possible, and so performance improves monotonically with the number of branches.

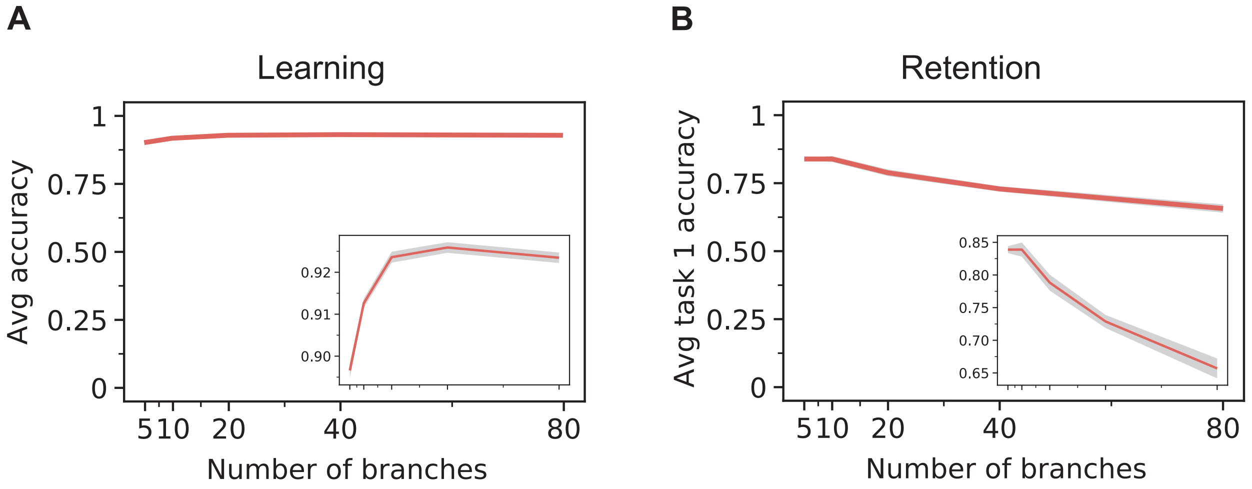

Figure S1: Permuted MNIST as a function of the number of DGN branches. **A**. Test accuracy at the end of training of each task, averaged over all 10 tasks. **B**. Test accuracy on task 1 after training on all 9 permutations. Grey areas are 99.5% confidence intervals of the results obtained from 10 models, initialised with different gating parameters and trained on differently permuted data.

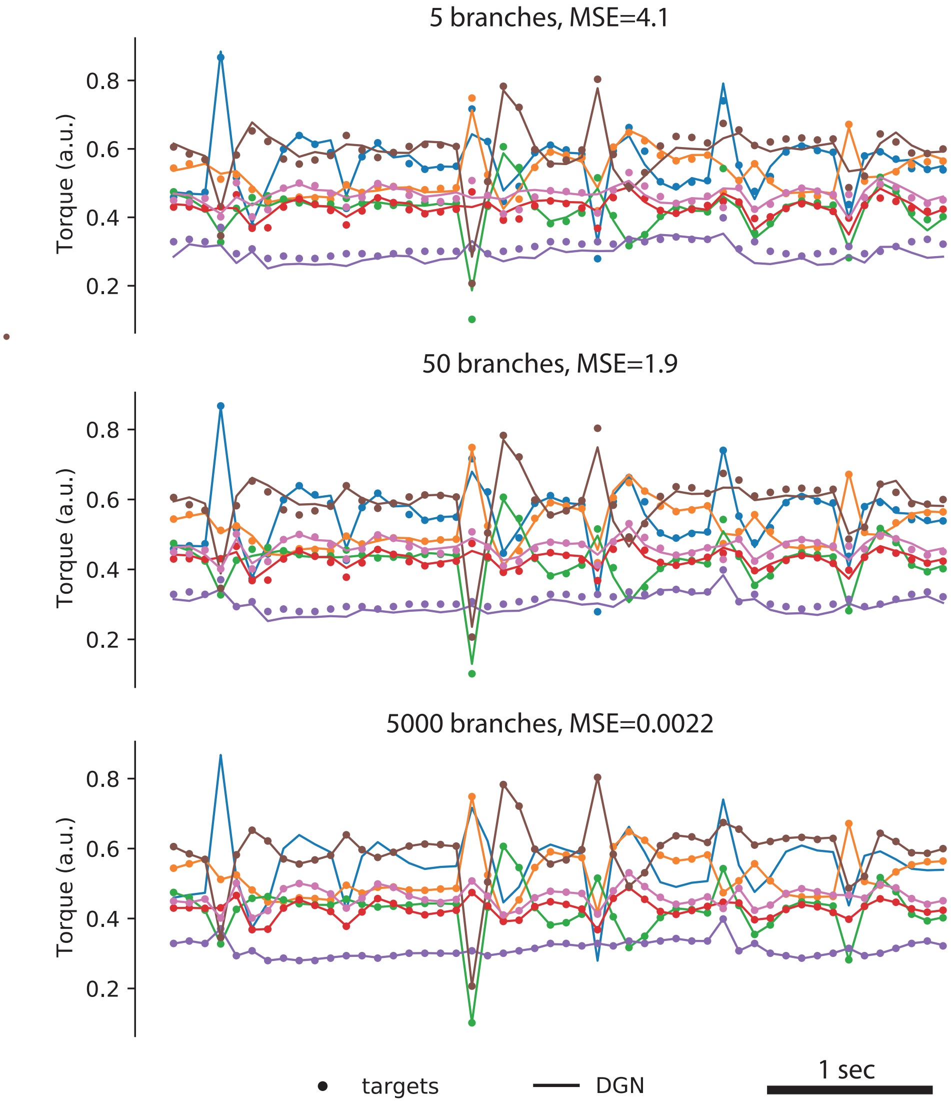

Figure S2: Sarcos solution for DGNs with 5, 50 and 5000 branches. Learning rate was 10^{-4} for 5 branches and 10^{-5} for 50 and 5,000 branches.

1002 Inverse Kinematics

1003 In Table S1 we compare the mean square error (MSE) obtained by DGN against
1004 baselines obtained from [31, 106, 107]. Note that, as mentioned in Methods, we (like
1005 others) used a test set that contained training examples.

Algorithm	MSE	Epochs
DGN	**0.002**	2000
Random forest	2.39	-
MLP	2.13	-
Stochastic decision tree	2.11	-
Gradient boosted tree	1.44	-
TabNet-S	1.25	55000
Adaptive neural tree	1.23	-
TabNet-M	0.28	55000
TabNet-L	0.14	55000
Gaussian Gated Linear Network	0.10	2000

Table S1: Test mean square error (MSE) and the number of passes over the dataset (i.e., number of epochs) for DGN with 5,000 branches versus previously published methods on the SARCOS inverse dynamics dataset [76, 106, 107]. DGN obtains the best result, by a factor of 50.

Catastrophic Forgetting (permuted MNIST)

Hyerparameter selection. We used a grid search to select the hyperparameters for the three networks (DGN, MLP and EWC). The parameters we tested are shown in Table S2; the ones that maximize test accuracy are in bold.

Model	learning rate	dropout	regularization const
DGN	10^{-4}, 10^{-3}, $\mathbf{10^{-2}}$, 10^{-1}	–	–
MLP	10^{-6}, 10^{-5}, $\mathbf{10^{-4}}$, 10^{-3}	Yes, **No**	–
EWC	10^{-6}, 10^{-5}, $\mathbf{10^{-4}}$, 10^{-3}	Yes, **No**	10^{2}, $\mathbf{10^{3}}$, 10^{4}

Table S2: For permuted MNIST, parameters swept during grid search. The best parameters (shown in bold) are the ones that maximize the average test accuracy over 20 random seeds.

Learning curves. In Fig. S3 we show the test performance of previously learned tasks (columns) as a function of the training across multiple tasks. To reduce clutter, a subset of the tasks (1, 2, 4, and 8, out of 10) are shown. The top left plot (train and test on task 1) shows that DGNs learn the first task much faster than all other methods. The plots to the right of that show retention on task 1 while the network is sequentially trained on subsequent tasks. MLP performances drop drastically after learning a few new tasks, while DGN and EWC show little forgetting. This is a remarkable feat for DGNs, which have no access to task boundaries and no explicit memory of previously learned tasks. EWCs, on the other hand, have both. If we look at the four diagonal plots, we see that DGN learns new tasks faster than all other methods, although the difference gets smaller as more tasks are learned.

The final accuracies across the diagonal correspond to the left panel of Fig. 4 whereas the final accuracies across the first row correspond to the right panel.

VOR

To obtain the smooth connectivity patterns seen in Fig. 7D, the initial weights had to be small. Larger initial weights produced non-smooth connectivity patterns, although the non-smoothness was different for MLPs than it was for DGNs. For MLPs, standard Glorot intialisation led to the noisy connectivity patterns shown in Fig. S4D, bottom panel; in contrast, to produce smooth patterns, the weights had to be scaled down by a factor of of 100. For DGNs, scaling the initial weights up by a factor of 10 relative to Fig. 7D produced noisy weights, but riding on a smooth background (Fig. S4D, top).

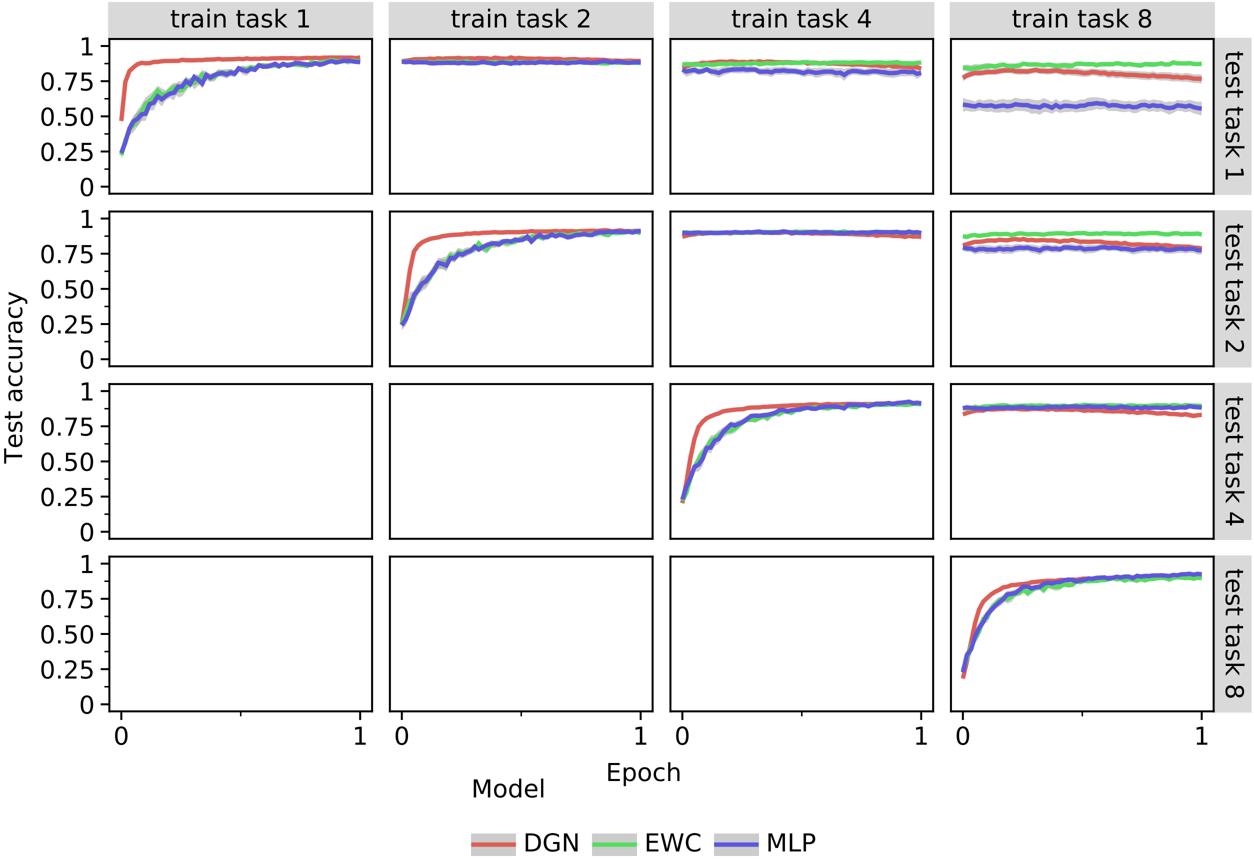

Figure S3: Retention results for permuted MNIST. Models are trained sequentially on 10 tasks, a subset of which is shown (tasks 1, 2, 4 and 8). Each column corresponds to a different stage of training (see labels on top), and each row reports test accuracy for a specific task. For example, the top row indicates performance on task 1 after being trained sequentially on tasks 1, 2, 4 and 8. Each model trains for one epoch per task; i.e., the 60,000 training examples per task are used only once. Error bars, indicated by the thickness of the lines, denote 95% confidence levels over 20 random seeds.

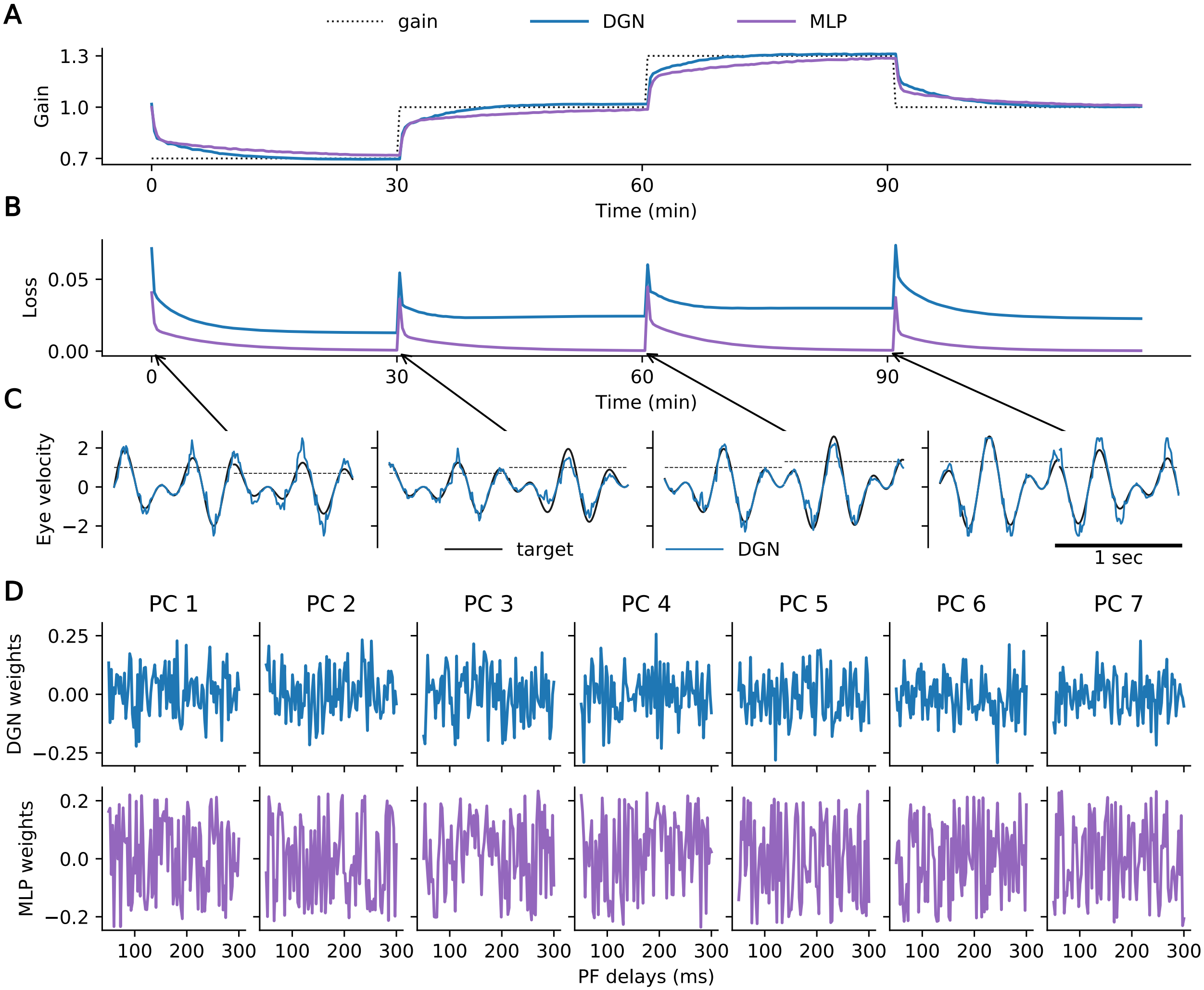

Figure S4: VOR network initialized with large, noisy weights. Same as Fig. 7, except that the training starts from large, noisy weights. For clarity, only five branches are shown in the top panel of **D** (compared to 10 in Fig. 7).

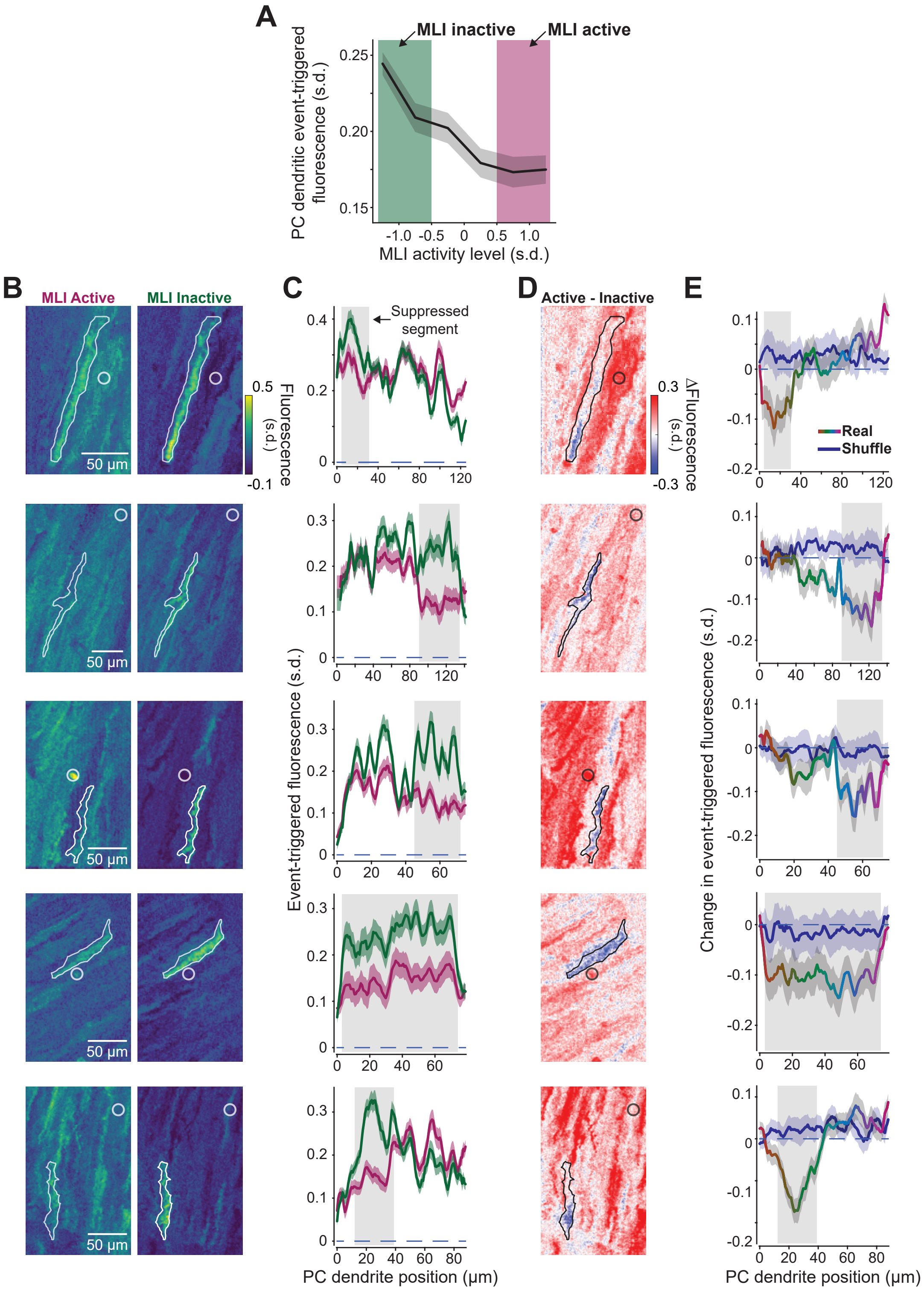

Figure S5: Suppression of Purkinje cell dendritic segments by MLIs. **A**. Event-triggered fluorescence in modulated Purkinje cell dendritic segments as a function of MLI activity level. Activity levels used for analysis in main figure are highlighted. **B**. Five additional examples of spatial event-triggered map of area surrounding Purkinje cell dendrites (contoured region of interest) when a nearby MLI (white circle; sometimes projected from different plane) was in an active state or inactive state. **C**. Spatial profile of event-triggered fluorescence of PC dendrites shown in panel A. **D**. Difference heatmap image of event-triggered fluorescence of same dendrites shown in panels A-B. **E**. Spatial profile trace (rainbow) and shuffled trace (blue) of of event-triggered difference heatmap images.

Pseudocode

Algorithm S1 DGN for quadratic loss

```
Input: network architecture: number of layers $K \in \mathbb{N}$,
       number of neurons in layer $k$ $\{n_k \in \mathbb{N}\}$,
       number of branches per neuron $i$ in layer $k$ $\{B_{k,i} \in \mathbb{N}\}$
Input: weights $\{w^b_{k,ij} \in \mathbb{R}\}$
Input: gating parameters $\{v^b_{k,ij} \in \mathbb{R}\}$, $\{\theta^b_{k,i} \in \mathbb{R}\}$
Input: input $\mathbf{x} = (x_1, ..., x_n) \in \mathbb{R}^n$
Input: target $r^* \in \mathbb{R}$
Input: learning rate $\eta \in (0, 1)$
Input: update $\in \{\textsc{true}, \textsc{false}\}$ (enables learning)
Output: Target prediction $\hat{r} = r_{K,1}$ (output of neuron in last layer $K$)
$r_{0,0} \leftarrow 1$; $n_0 \leftarrow n$; $r_{0,i} = x_i$ for $i \in \{1, ..., n\}$
for $k \in \{1, \ldots, K\}$ do {over layers}
    $r_{k,0} \leftarrow 1$ {bias}
    for $i \in \{1, \ldots, n_k\}$ do {over neurons}
        for $b \in \{1, \ldots, B_{k,i}\}$ do {over branches}
            $g^b_{k,i} \leftarrow \Theta(\sum_{j=0}^{n_{k-1}} v^b_{k,ij} x_j - \theta^b_{k,i})$
        $r_{k,i} \leftarrow \sum_{b=1}^{B_{k,i}} g^b_{k,i} \sum_{j=0}^{n_{k-1}} w^b_{k,ij} r_{k-1,j}$
        if update then
            for $b \in \{1, \ldots, B_{k,i}\}$ do {over branches}
                if $g^b_{k,i} > 0$ then
                    for $j \in \{1, ..., n_{k-1}\}$ do {over neurons in previous layer}
                        $w^b_{k,ij} \leftarrow w^b_{k,ij} - \eta\,(r_{k,i} - r^*)\, w^b_{k,ij} r_{k-1,j}$
return $r_{K,1}$
```

Here $\Theta(\cdot)$ is the Heaviside step function ($\Theta(z) = 1$ for $z > 0$ and $\Theta(z) = 0$ otherwise).

Algorithm S2 DGN for Bernoulli data

```
Input: network architecture: number of layers $K \in \mathbb{N}$,
       number of neurons in layer $k$ $\{n_k \in \mathbb{N}\}$,
       number of branches per neuron $i$ in layer $k$ $\{B_{k,i} \in \mathbb{N}\}$
Input: weights $\{w^b_{k,ij} \in \mathbb{R}\}$
Input: gating parameters $\{v^b_{k,ij} \in \mathbb{R}\}$, $\{\theta^b_{k,i} \in \mathbb{R}\}$
Input: precision $\epsilon \in (0, 0.5)$
Input: input $\mathbf{x} = (x_1, ..., x_n) \in \mathbb{R}^n$
Input: target $r^* \in \{0, 1\}$
Input: learning rate $\eta \in (0, 1)$
Input: update $\in \{\textsc{true}, \textsc{false}\}$ (enables learning)
Output: Target prediction $\hat{r} = r_{K,1}$ (output of neuron in last layer $K$)
$r_{0,0} \leftarrow \sigma(1)$; $n_0 \leftarrow n$; $r_{0,i} = \textsc{clip}^{1-\epsilon}_{\epsilon}(\sigma(x_i))$ for $i \in \{1, ..., n\}$
for $k \in \{1, \ldots, K\}$ do {over layers}
  $r_{k,0} \leftarrow \sigma(1)$ {bias}
  for $j \in \{1, \ldots, n_{k-1}\}$ do {over neurons in layer below}
    $h_{k-1,j} \leftarrow \sigma^{-1}(r_{k-1,j})$
  for $i \in \{1, \ldots, n_k\}$ do {over neurons}
    for $b \in \{1, \ldots, B_{k,i}\}$ do {over branches}
      $g^b_{k,i} \leftarrow \Theta(\sum_{j=0}^{n_{k-1}} v^b_{k,ij} x_j - \theta^b_{k,i})$
    $h_{k,i} \leftarrow \sum_{b=1}^{B_{k,i}} g^b_{k,i} \sum_{j=0}^{n_{k-1}} w^b_{k,ij} h_{k-1,j}$
    $r_{k,i} \leftarrow \textsc{clip}^{1-\epsilon}_{\epsilon} \sigma(h_{k,i})$
    if update then
      for $b \in \{1, \ldots, B_{k,i}\}$ do {over branches}
        if $|r^* - \sigma(h_{k,i})| > \varepsilon$ then
          for $j \in \{1, ..., n_{k-1}\}$ do {over neurons in previous layer}
            $w^b_{k,ij} \leftarrow w^b_{k,ij} - \eta(r_{k,i} - r^*) h_{k-1,j}$
return $r_{K,1}$
```

Here, as above, $\textsc{clip}^b_a(\cdot)$ clips values between a and b,

$$\textsc{clip}^b_a(y) \equiv \begin{cases} a & y < a \\ y & a < y < b \\ b & b \le y \end{cases} \quad . \tag{16}$$

Also as above, $\sigma(\cdot)$ is the sigmoid function, $\sigma(z) = e^z/(1 + e^z)$. Its inverse is given by $\sigma^{-1}(y) = \log(y/(1 - y))$.