Towards a More General Understanding of the Algorithmic Utility of Recurrent Connections

Brett W. Larsen[1-3] (bwlarsen@stanford.edu), Shaul Druckmann[2,3] (shauld@stanford.edu)

1. Department of Physics, Stanford University, Stanford, CA
2. Department of Neurobiology, Stanford University School of Medicine, Stanford, CA
3. Wu Tsai Neurosciences Institute, Stanford University, Stanford, CA

Abstract

Lateral and recurrent connections are ubiquitous in biological neural circuits. The strong computational abilities of feedforward networks have been extensively studied; on the other hand, while certain roles for lateral and recurrent connections in specific computations have been described, a more complete understanding of the role and advantages of recurrent computations that might explain their prevalence remains an important open challenge. Previous key studies by Minsky and later by Roelfsema argued that the sequential, parallel computations for which recurrent networks are well suited can be highly effective approaches to complex computational problems. Such "tag propagation" algorithms perform repeated, local propagation of information and were introduced in the context of detecting connectedness, a task that is challenging for feedforward networks. Here, we advance the understanding of the utility of lateral and recurrent computation by first performing a large-scale empirical study of neural architectures for the computation of connectedness to explore feedforward solutions more fully and establish robustly the importance of recurrent architectures. In addition, we highlight a tradeoff between computation time and performance and demonstrate hybrid feedforward/recurrent models that perform well even in the presence of varying computational time limitations. We then generalize tag propagation architectures to multiple, interacting propagating tags and demonstrate that these are efficient computational substrates for more general computations by introducing and solving an abstracted biologically inspired decision-making task. More generally, our work clarifies and expands the set of computational tasks that can be solved efficiently by recurrent computation, yielding hypotheses for structure in population activity that may be present in such tasks.

Author Summary

Lateral and recurrent connections are ubiquitous in biological neural circuits; intriguingly, this stands in contrast to the majority of current-day artificial neural network research which primarily uses feedforward architectures except in the context of temporal sequences. This raises the possibility that part of the difference in computational capabilities between real neural circuits and artificial neural networks is accounted for by the role of recurrent connections, and as a result a more detailed understanding of the computational role played by such connections is of great importance. Making effective comparisons between architectures is a subtle challenge, however, and in this paper we leverage the computational capabilities of large-scale machine learning to robustly explore how differences in architectures affect a network's ability to learn a task. We first focus on the task of determining whether two pixels are connected in an image which has an elegant and efficient recurrent solution: propagate a connected label or tag along paths. Inspired by this solution, we show that it can be generalized in many ways, including propagating multiple tags at once and changing the computation performed on the result of the propagation. To illustrate these generalizations, we introduce an abstracted decision-making task related to foraging in which an animal must determine whether it can avoid predators in a random environment. Our results shed light on the set of computational tasks that can be solved efficiently by recurrent computation and how these solutions may appear in neural activity.

Introduction

One of the brain's most striking anatomical features is the ubiquity of lateral and recurrent connections. Though this feature has been known since the earliest anatomical investigations, a theoretical understanding of the utility of recurrent and lateral connections has yet to be established, at least partially due to the difficulty of the question. Intriguingly, the ubiquity of recurrent connections is perhaps the architectural aspect most divergent from the majority of contemporary artificial neural network research, including the successful class of algorithms collectively referred to as deep learning. Indeed, with enough layers and neurons any task can be performed by a feedforward network [1-4], and even single layer, purely feed-forward networks have powerful computational capabilities [5]. While architectures with recurrent connections are used in specific applications, typical deep learning architectures are purely feedforward and lack any lateral or recurrent connections. This raises the possibility that part of the difference in computational capabilities between real neural circuits and artificial neural networks is accounted for by the role of recurrent connections. For these reasons, a more detailed understanding of the computational role played by lateral and recurrent connections is of great importance.

From the neuroscience perspective, lateral and recurrent connections have been hypothesized to play an important role in diverse functions such as divisive normalization [6], predictive coding [7, 8], and contextual interactions such as contour detection [9, 10]. From the machine learning perspective, recurrent networks are typically used when modeling or parsing temporal sequences, such as vocal time series or strings of words in a sentence [11-15]. More generally, in their famous discourse on perceptrons, Minsky and Papert provided concrete examples of tasks in which feedforward networks are inefficient, but parallel, sequential approaches, akin to RNNs are very effective [16]. Briefly stated, they demonstrated that certain computations that are global functions of the input are extremely inefficient to implement in a single layer perceptron. An early example was the parity function on binary variables, which has a positive output if the number of positive inputs is odd, where changing any one of the inputs will change the output.

Minsky and Papert then showed that some more biologically relevant computations also suffer from the same inefficiency. A key example was the computation of connectedness, namely determining whether two points are connected by pixels of similar color or texture (**Fig. 1A**). It is intuitive that this is also a global computation since the connectedness property between two objects can be altered by toggling one or a few pixels throughout the image (**Fig. 1A**). In a series of seminal papers, Roelfsema and colleagues [17-21] demonstrated and studied a solution based on sequential, parallel computation networks (very similar to what are currently referred to as vanilla RNNs), a solution which they refer to as tag propagation. They demonstrated that it is substantially more efficient than particular feedforward solutions that they studied.

Though tag propagation is clearly an efficient solution to the detection of connectedness, the comparison to feedforward solutions, and particularly to deep feedforward networks was computationally limited because it predated hardware and software improvements in training artificial neural networks [22]. In addition, while connectedness is an important computation that is central to visual perception, lateral and recurrent connections are so ubiquitous in the brain that it is natural to seek a more general understanding of task structures which are well-suited to architectures with lateral and recurrent connections. Here, we advance the understanding of the utility of lateral and recurrent computation by first performing a detailed large-scale empirical study of neural architectures for the computation of connectedness to explore feedforward solutions more fully and establish robustly the importance of recurrent architectures. We then begin to generalize our understanding of the tasks that may benefit from recurrent connections by demonstrating that tag-propagation like solutions can be an efficient computational substrate for more general tasks. Namely, we present and study a task which we call the predator-prey task in which an animal needs to make a decision to hide from predators or attempt to run to shelter. The solution to this task is based on multiple extensions to tag-propagation, which we refer to as generalized tag propagation and further highlights how such tag-like computations can distill global information necessary for a computation into local information.

Results

Edge-Connected Pixel Task Description

A fundamental task of the visual system is recognizing objects, and a key attribute of most objects is that their subparts are connected. That is, two pixels in an image corresponding to the same object will usually have similar color, texture, or other features and are connected by a path of pixels that share these visual attributes. Here, we tested the ability of different neural architectures to recognize connectedness. To simplify, we consider binary images where connectedness can be defined precisely: given two pixels in the image, they are connected if there is a path between them in the image of positive pixels (**Fig. 1A)**. To investigate the detection of connectedness while avoiding dependence on specific object geometry, we study a task whose goal is to label all pixels connected to the edge of the image, which we refer to as the edge-connected pixel task (**Fig. 1B**). This can be thought of as performing the connectedness task many times in parallel for all the edge pixels. More specifically, we used binary $N \times N$ images $\boldsymbol{I} = \{-1, +1\}^{N\times N}$ such that the image has N^2 pixels and we refer to +1 pixels as “on” and -1 pixels as “off.” We defined a pixel's connectivity according to its four-facing nearest neighbors, i.e., paths were not considered diagonally between pixels. Thus, we define the overall edge-connected pixel task as taking a binary image $\boldsymbol{I}$ as input and returning an image that labels all pixels that are on and connected to the edge of the image as +1, i.e., edge-connected or just connected, and all other pixels as -1, i.e., edge-disconnected or just disconnected (**Fig. 1B**).

Similar to how pixel-pair connectivity is computed, the edge-connected task would be extremely inefficient to solve by enumerating all possible paths since the number of paths grows exponentially with image size (**Fig. 1C**). Akin to the solution suggested by Roelfsema for pixel pair connectivity [17, 18], the task can instead be solved efficiently by a tag-propagation strategy. In an architecture with number of units equal to the number of image pixels in which each unit describes a pixel for each input image, we first set the initial labels to be +1 (connected) for all the edge pixels for which the input is +1 (on) and label the remaining pixels as -1 (disconnected). We then propagate these labels by sequentially setting labels to +1 (connected) if their input is +1 (on) and they have a nearest neighbor that is labeled as +1 (connected). This sequential propagation continues until convergence, i.e., an update does not change the condition of any label (**Fig. 1D**).

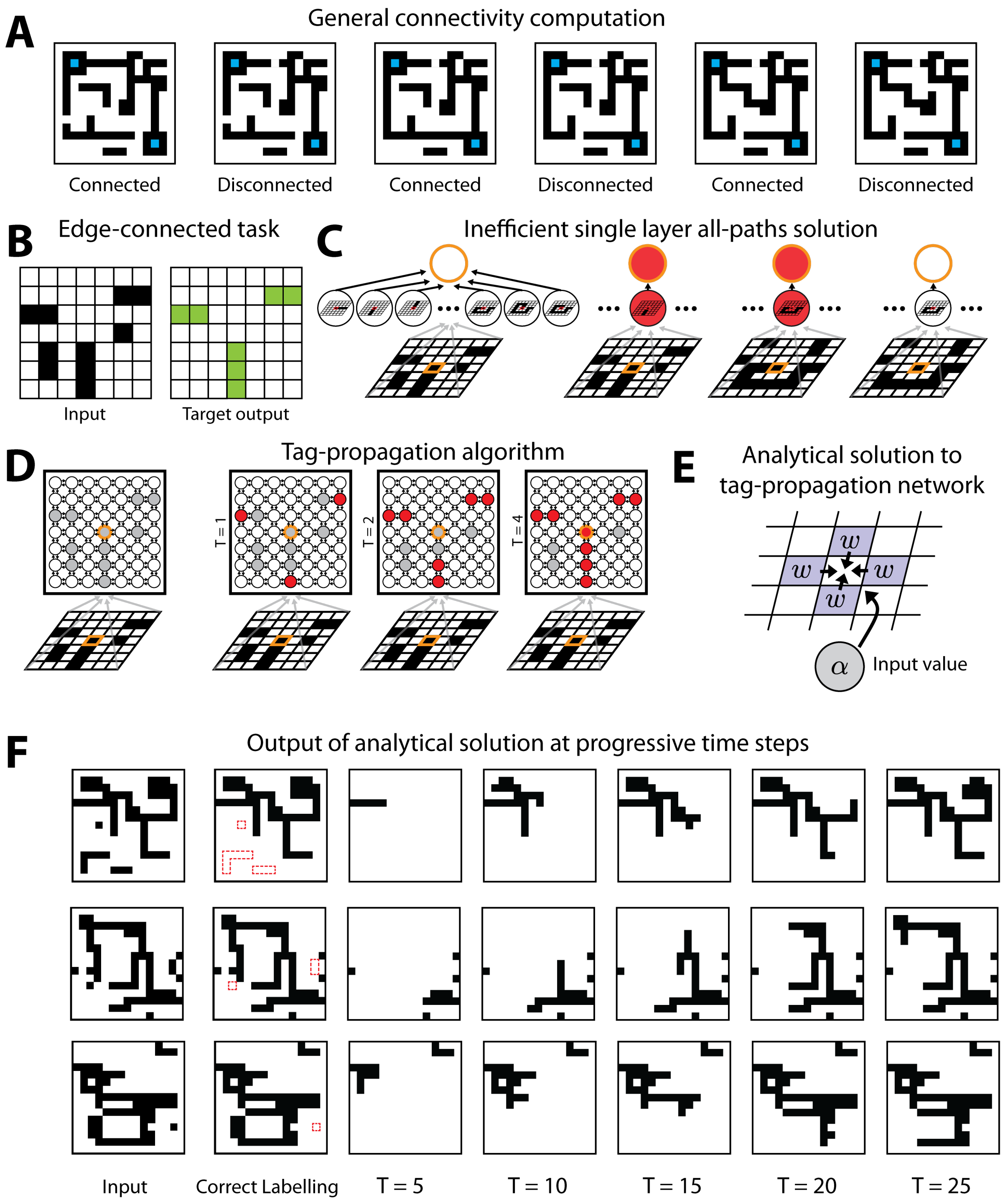

A
General connectivity computation
Connected
Disconnected
Connected
Disconnected
Connected
Disconnected
B
Edge-connected task
Input
Target output
C
Inefficient single layer all-paths solution
D
Tag-propagation algorithm
T = 1
T = 2
T = 4
E
Analytical solution to tag-propagation network
w
α
Input value
F
Output of analytical solution at progressive time steps
Input
Correct Labelling
T = 5
T = 10
T = 15
T = 20
T = 25

Fig 1. Edge-connected pixel task. (A) Illustration of the general connectivity task and its global nature. (B) Example of an input image and correct labelling for that image in the edge-connected pixel task. (C) Implementation of the single-hidden layer architecture for detecting whether the center pixel is connected to the edge. Each hidden neuron checks whether a certain pattern connecting the pixel to the edge is in the image. (D) Tag propagation solution in a recurrent network for detecting whether the center pixel is connected to the edge. We start with any ON pixels connected to the edge tagged as connected. At subsequent time steps, the tag is passed to any neighboring pixel which is also ON. (E) Schematic of the setup for the analytical solution of implementation of the tag propagation algorithm in neural network weights. The same setup is repeated at all pixels in the image. (F) Output of the analytical tag-propagation network at progressive time steps for several example images.

We first showed analytically that the standard formulation of a recurrent network can implement a tag-propagation solution. In the tag propagation approach, we start from an initial set of activated pixels and activate pixels sequentially if they are connected to an active pixel. For simplicity of comparison, let us assume that the recurrent network has as many neurons as pixels and that each neuron corresponds to a specific pixel, i.e., it receives input-image connections only from the single, corresponding pixel in the input image. Accordingly, the state of the recurrent network at time t, $\mathbf{x}^{(\mathrm{t})}$, is a vector of dimensionality equal to the number of pixels. The dynamics of the network are given by:

$$\mathrm{x}_{\mathrm{i}}^{(\mathrm{t})} = \tanh\left(\beta\left[\left(\sum_{\mathrm{j}} \mathrm{W}_{\mathrm{ij}}\mathrm{x}_{\mathrm{j}}^{(t-1)}\right) + \upalpha_{\mathrm{i}}\mathrm{I}_{\mathrm{i}} + \mathrm{b}_{\mathrm{i}}\right]\right) \quad (1)$$

where W_{ij} are weights from other neurons, α_i are weights from the input image and b_i is a bias term (**Fig. 1E**). Since in the tag propagation solution the next time step depends only on the value of the input pixel corresponding to the *i*-th unit and the current value of the *i*-th units neighboring pixels, we can reduce the incoming weights for each neuron to five weights: four weights W_{ij} for j corresponding to the nearest neighbor pixels in the image and the input-image weight α_i. Together with the bias parameter b_i this yields six parameters per neuron. Furthermore, we can take advantage of an additional symmetry. Each of the neighbors should have the same effect on a neuron's state: if any of them has a connected tag and the pixel is on, then it should also have a connected tag in the next step. Thus, the four W_{ij} can be reduced to a single w_i that is shared among the four pixels. Finally, as the same computation should be performed at each neuron, we can have the weights identical for all neurons, i.e., we drop the dependency on the neuron index. Taken together, the conditions for these weights to perform the tag propagation correctly are therefore given by a set of three inequalities:

$$\begin{aligned} b - 2w + \alpha &> 0 \\ b - 4w + \alpha &< 0 \\ b + 4w - \alpha &< 0 \end{aligned} \quad (2)$$

The first equation expresses the condition that a pixel should become activated if its input is on and if at least one neighboring pixels is on (i.e., the three others are off, contributing $1 \times (+1) \times w + 3 \times (-1) \times w = -2w$). The second equation expresses the condition that a neuron should be off if its input pixel is on yet all its neighboring pixels are off (contributing $4 \times (-1) \times w = -4w$). The third equation expresses the condition that a pixel should not be activated even if all its neighbors are on (contributing $4 \times (+1) \times w = 4w$) but its input pixel is off. Note that the above inequalities hold specifically for pixels not on the edges or in the corners. In these other locations, the inequalities need to be modified to account for there being only 3 and 2 neighbors respectively. Moreover, these are based on a single step. If the neurons are binary, as in the original tag propagation proposal, or if β is high, the same solution holds for the full series of time steps.

Simulating these analytically derived weights, we find the network reaches zero error if run for a sufficient number of time steps (**Fig. 1F**), as expected.

Having established the tag propagation solution in recurrent architectures, we systematically explored how multiple different types of architectures solve the edge-connected pixel task by training networks to output the correct edge-connected classification of training examples by stochastic gradient descent, SGD (**Fig. 2A** and Methods). For each architecture, we considered networks with various numbers of layers, from only a few layers, which should be easily trainable but potentially less powerful, up to thirty layers. We note that––generally speaking—despite deep learning's great success, training networks by stochastic gradient descent is not guaranteed to find optimal solutions [23]. To ameliorate this issue, we first performed extensive searches on the space of learning parameters, e.g. learning rates and batch size a process known as hyperparameter optimization (see Methods). To further ensure finding successful models, we performed hyperparameter optimization separately for each architecture type and network depth (**Fig. 2B** and Methods). Second, after we discovered the best learning parameters based on the learned network performance, we trained a large number of networks from independent initializations since starting from distinct initial conditions may lead to convergence to a different local optimum (**Fig. 2B** and Methods). We chose to study the best solutions from these independently initialized networks.

The first network architecture we trained was a deep feedforward architecture. This architecture receives the image as input to the first layer and is composed of multiple hidden units in each of several layers. The output was inferred from the activation of units in the last layer, referred to as the output layer (**Fig. 2A**). Its input-output function is given by:

$$\mathbf{y} = \sigma(\mathbf{W}^L(\dots\sigma(\mathbf{W}^1\mathbf{x} + \mathbf{b}^1)\dots) + \mathbf{b}^L) \tag{3}$$

Here, σ is a tanh function, $\mathbf{x}$ is the input image, the matrix $\mathbf{W}^1$ is the weights from the input to the first layer, $\mathbf{W}^L$ the weights from one layer to the next and $\mathbf{b}^L$ is a bias term. We note that in following equations we will drop the explicit notation of the constant bias term to simplify presentation, absorbing it in the standard way into the weights by adding a fixed input to all training examples ([24]; see Methods). We set the size of the hidden layers to be the size of the input $\mathbf{x}$. Thus, the network had $L(N^2 \times N^2)$ parameters for the fully connected linear weights $\{\mathbf{W}^1, \dots, \mathbf{W}^L\}$ and LN^2 parameters for the biases $\{\mathbf{b}^1, \dots, \mathbf{b}^L\}$ for a total of $L(N^4 + N^2)$ parameters. This network architecture is illustrated in **Figure 2B**. Note that due to the common problem of vanishing gradients in deep feedforward models, a small number of skip connections were added from the input to deeper layers for models with 15-30 layers, which we found improved network performance (see Methods).

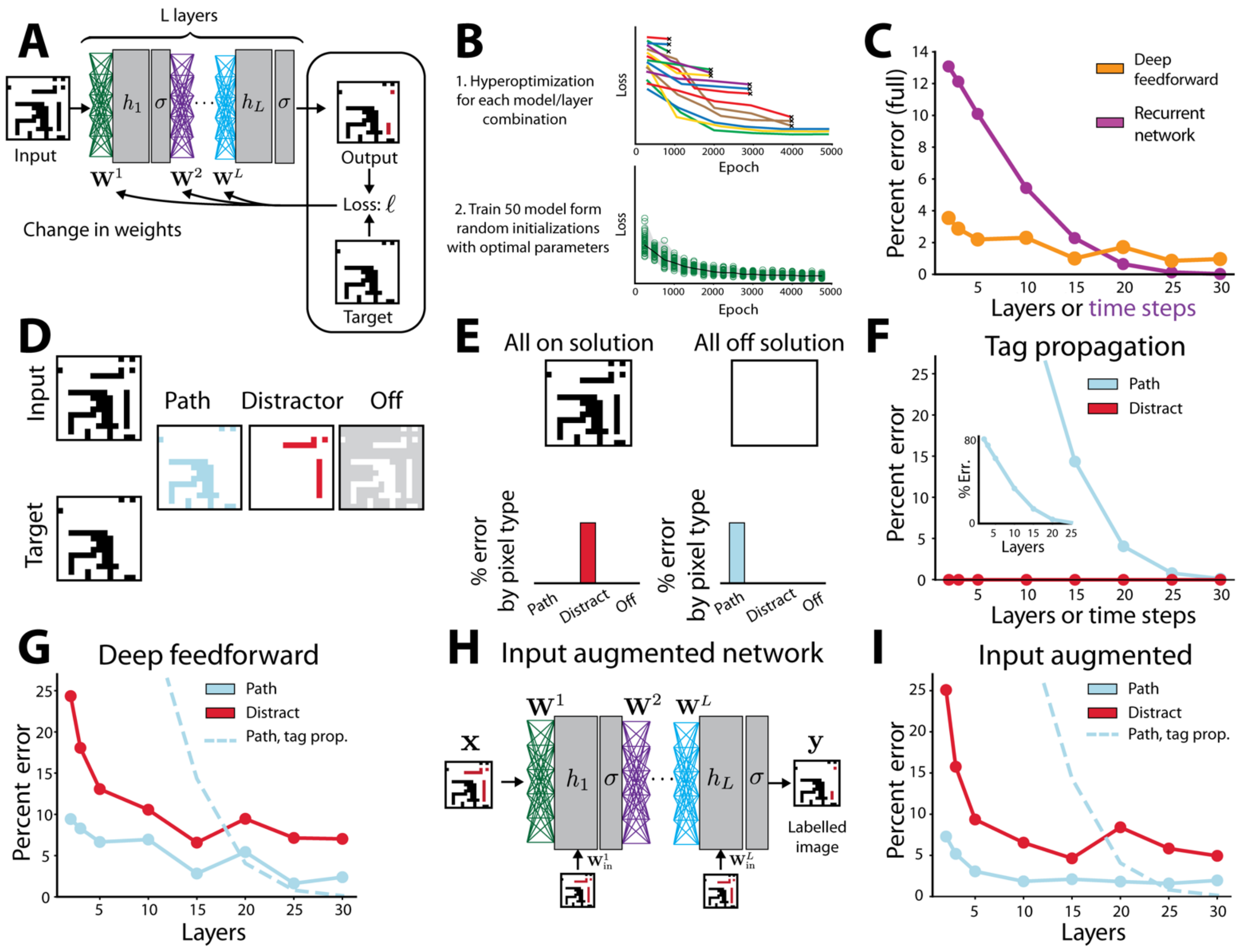


Fig 2. Feedforward network performance on the edge-connected pixel task. (A) Training procedure for neural networks. Weights are learned iteratively. At each iteration the weights are updated using the gradient of the loss function calculated via backpropagation. The network illustrated here is a deep feedforward network. The image is given as input to the first layer and the activation of units at the last layer is the network's output. (B) Schematic of the two-part procedure used to obtain good solutions despite the stochasticity of learning and the large parameter spaces. First, hyperoptimization is performed over the training parameters for each model and layer combination (top). Each line corresponds to a network being trained, colored differently to indicate the different hyperparameters used. The worst performing models are eliminated at regular intervals (see Methods for full details). Second (bottom), 50 models are trained from random initialization, all using the optimal parameters found in the hyperoptimization procedure. (C) Performance of the best trained solution for the deep feedforward network across layers vs. the recurrent tag propagation solution. X-axis corresponds to number of layers in the network for the feedforward solution, and number of time-steps the network is allowed to run for the recurrent solutions, which have only one layer of recurrently connected neurons. (D) Illustration of the splitting of pixels, and associated errors, into three groups: path, distractor, and off. (E) Breakdown of errors for two naïve solutions: All on, outputting all the on-pixels as connected, and all off, outputting no pixels as connected. (F-G) Decomposition of error by pixel type for each model. (F) Tag propagation implemented via recurrent network. X-axis corresponds to the number of time-steps the network is allowed to run. Blue dots and line correspond to errors on path pixels, red dots and line correspond to errors on distractor pixels. Inset shows same data with larger axis range. (G). X-axis corresponds to number of layers in the network. Blue dots and line correspond to errors on path pixels, red dots and line correspond to errors on distractor pixels. Dashed line corresponds to tag propagation error on path pixels for reference. (H) Schematic of input augmented architectures where the full image is added as input at each layer, allowing the tag propagation solution to be part of parameter space. (I) Decomposition of error by pixel type for the input augmented network. Same plotting convention as G.

We began by measuring the raw accuracy of trained networks. For each layer number we trained 50 networks from distinct random initializations using layer-specific hyperparameters for 5000 epochs and displayed the best network's performance on a held-out dataset (**Fig. 2C**). We evaluated error by each network by outputting an $N \times N$ image that predicts for each pixel whether it should be labelled as connected. Error decreased with increased layer number, indicating we successfully trained networks even when they had many layers. Moreover, we found that deep feedforward networks were able to find approximately accurate solutions even with hidden layers of a size no larger than the number of pixels, a solution far more efficient than the exponential scaling of the straightforward solution of exhaustive enumeration in a single hidden layer previously suggested (**Fig. 2C**). To better understand the solutions found by the deep neural network we compared errors to those of the tag-propagation solution. The tag-propagation solution, if allowed to run till convergence, will yield zero error on all types of pixels. If stopped after a fixed number of iterations, however, it will only correctly label connected pixels close to the edge, yielding non-zero error (**Fig. 2C**). We found that if time is a significant limitation, meaning that network architectures can only propagate for a few time steps or through a small number of layers, then the feedforward solutions learns far more accurate solutions (**Fig. 2C**). With just two layers, the deep feedforward network was able to learn solutions with 3.54% error vs. 13.1% error in the tag propagation solution.

Before analyzing the error in more detail, however, we noted that accuracy on all pixels is not the most informative measure because it conflates multiple types of error. For instance, pixels that are off in the input will never be connected, and labeling all pixels that were off as unconnected is an easy way to reduce error on off-pixels to zero. Therefore, to obtain more informative metrics, we subdivided the error by mutually exclusive pixel types (**Fig. 2D**). The first group of pixels are "path" pixels which are on and connected to the edge by other on-pixels. An error on these pixels indicates a failure to identify all connected pixels. The next were "distractors": on-pixels disconnected from the edge, which should be labelled as disconnected, and "off" pixels which are off in the image and should all be labelled as disconnected. As an intuition for this breakdown of pixels and error type, a network that simply labeled all pixels as connected if their value in the input image was on, and disconnected if their value in the input image was off would score zero error in path pixels, zero error in the non-activated pixels but a full error (i.e., 100 percent error) in the distractor pixels (**Fig. 2E**). Conversely, a network that labels all pixels as disconnected would yield zero error in distractor and disconnected pixels and a full error in the path pixels (**Fig. 2E**).

The relative number of pixels in these categories depended on the way input images were generated. Intuitively, the two challenging aspects of the task are long paths and the presence of distractors. As expected, we found that the two most important properties that increase task difficulty were the presence of extended-length paths connected to the edge of the image and the percentage of distractor pixels. Images were generated by seeding pixels randomly and continuing a path randomly based on these seeds (see Methods). Using this sampler generation method, images had on average 15.9% of their pixels that were on and connected to the edge and 8.3% of the pixels that were on and disconnected from the edge, i.e., distractors, with the remaining 75.8% pixels off. Other choices of sample generation yielded different ratios of pixel types and qualitatively similar results across architectures as long as they contained a substantial number of challenging examples.

By design, the tag propagation networks were guaranteed to make no errors in non-activated pixels or distractor pixels (**Fig. 2F**). We found that deep networks learn different forms of solutions, producing a mixture of errors on path and distractor pixels (**Fig. 2G**). This was still true even for deep learning networks with twenty or more layers, by which point the tag propagation solutions became more effective. The feedforward networks as defined above are the most natural extension of previously studied single hidden layer networks and the standard architecture of fully connected networks in deep learning. As formulated, however, there was an important mismatch between them and the tag propagation architecture. Namely, although the input can filter through the layers, it only directly affects the bottom layer; in recurrent networks (and the tag propagation solution), the network interacts with the input at each time step. To address this issue, we trained an additional network architecture, which we refer to as input-augmented, where each layer also receives the original image as input (**Fig. 2H**). The input-output function for such architectures is given by:

$$\mathbf{y} = \sigma(\mathbf{W}^L \ldots \sigma(\mathbf{W}^1\mathbf{x} + \mathbf{W}_{\text{in}}^1\mathbf{I}) + \mathbf{W}_{\text{in}}^L\mathbf{I}) \tag{4}$$

where $\mathbf{W}_{\text{in}}^k$ are the weights from the input to the hidden units of the kth layer, each of which is $N^2 \times N^2$. Thus, the total number of parameters in the network is $L(N^4 + N^4 + N^2) = L(2N^4 + N^2)$ parameters.

The input-augmented networks slightly outperformed the standard feedforward networks both in terms of overall performance and the number of layers needed to achieve it (**Fig. 2I**). The best overall performance by the deep feedforward network was 1.63% error on path pixels and 7.14% error on distractor pixels (an average of 3.52% error, not counting off pixels) at 25 layers. Meanwhile, the input-augmented network achieved its best performance with 2.10% error on path pixels and 4.63% error on distractor pixels (an average of 2.97% error not counting off pixels) at 15 layers. Both architectures still underperformed relative to the recurrent network solutions, even though the tag propagation solution was in their parameter space. In other words, we know that a solution with close to 0% error could have been learned by the input-augmented network, but the error breakdown showed that this solution was not learned.

In summary, even with the computational resources of modern deep learning, the purely feedforward networks we trained did not achieve the same task performance as the simple recurrent tag propagation network. On the other hand, we found that feedforward solutions did not converge on naïve poor-performing solutions, such as attempting to enumerate an exponential number of solutions. Instead, they were able to learn compromise solutions that were more effective than tag-propagating in the low layer or timestep regime, demonstrating the importance of empirically studying solutions by deep learning.

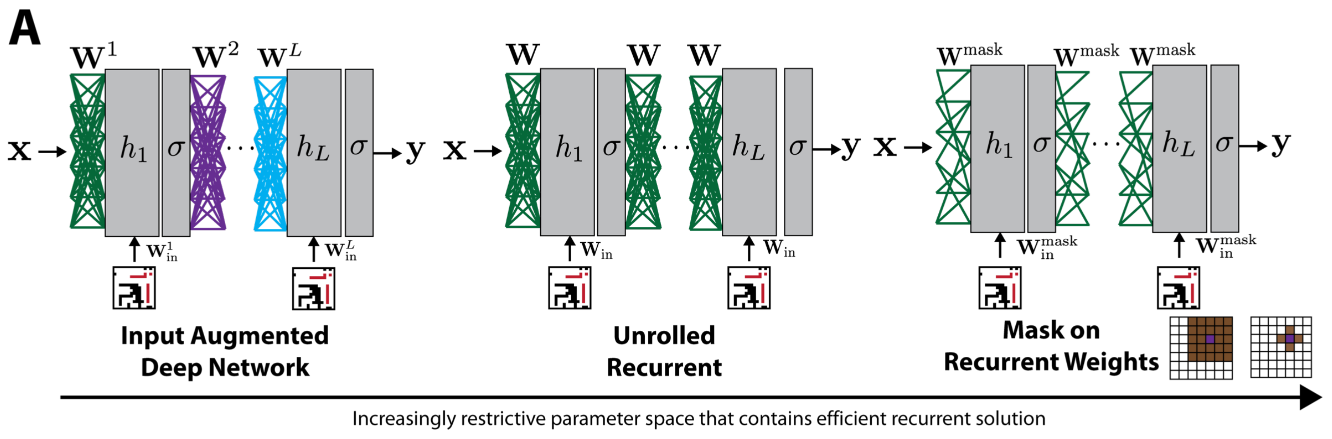
Networks with Weight Sharing

To better understand why the tag-propagation solution was not learned despite being an achievable solution of the input-augmented networks, we tested three new network architectures that are gradated reductions of the solution space. These still contain the tag-propagation solution, but with an order of magnitude fewer synapses (**Fig. 3A**). Each reduction was motivated by an invariance or a locality in the tag-propagated solution.

The first invariance we considered in the tag-propagated solution was the invariance across time. Each time step performs the same calculation, and thus we first consider a network with weights shared across layers (i.e., limiting the network to only one matrix $\boldsymbol{W}$ between hidden layers, one matrix of input weights $\mathbf{W}_{\text{in}}$, and a single vector of biases $\boldsymbol{b}$). The input-output function of this architecture is given by:

$$\mathbf{y} = \sigma(\mathbf{W} \ldots \sigma(\mathbf{Wx} + \mathbf{W}_{\text{in}}\mathbf{I}) + \mathbf{W}_{\text{in}}\mathbf{I}) \quad (5)$$

As before, $\boldsymbol{b}$ has been absorbed into the weights. Such a solution is sometimes referred to as a "rolled-out" recurrent network, since the input output function is identical to a recurrent network running for a number of time steps equal to the number of layers (L) driven by a constant (in time) input $\mathbf{I}$. This invariance causes the number of parameters to be substantially smaller and not to scale with the depth of the network; the total number of parameters in the network is $N^4 + N^4 + N^2 = 2N^4 + N^2$.


A
$\mathbf{W}^1$
$\mathbf{W}^2$
$\mathbf{W}^L$
$\mathbf{x}$
h_1
σ
h_L
$\mathbf{y}$
$\mathbf{W}^1_{\text{in}}$
$\mathbf{W}^L_{\text{in}}$
Input Augmented Deep Network
$\mathbf{W}$
$\mathbf{W}_{\text{in}}$
Unrolled Recurrent
$\mathbf{W}^{\text{mask}}$
$\mathbf{W}^{\text{mask}}_{\text{in}}$
Mask on Recurrent Weights
Increasingly restrictive parameter space that contains efficient recurrent solution

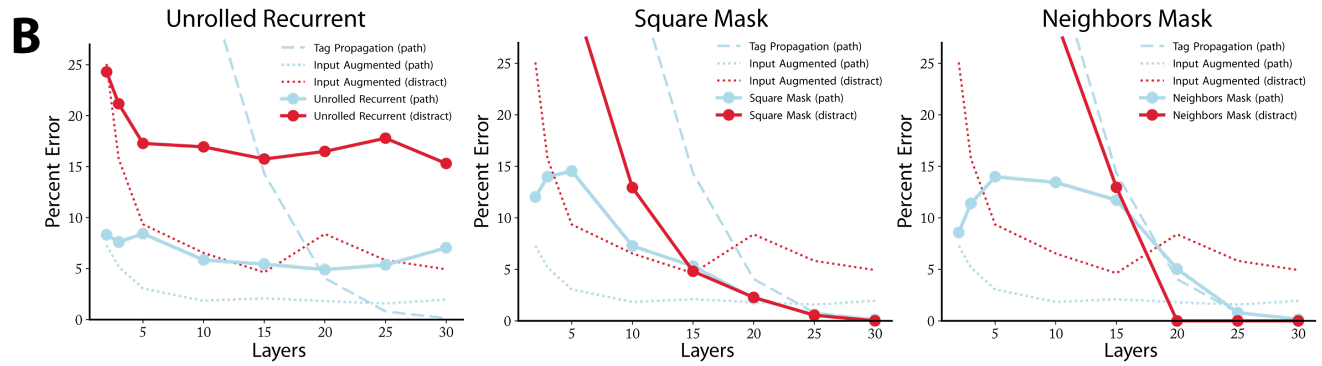


Fig 3. Masked recurrent networks performance on the edge-connected pixel task. (A) Schematic of the addition of constraints on the feed-forward parameter space generating an increasingly restrictive parameter space that still contains the efficient tag-propagation solution. From left to right: weight sharing across the layers creates an unrolled recurrent network, masking of shared weights (i.e., enforcing locality in the operations) to a grid around each pixel creating unrolled recurrent networks with sparse weights (middle), masking to just the nearest neighbor of each pixel (right). (B) Decomposition of error by pixel type for each model. X-axis corresponds to number of layers in the network. Blue dots and line correspond to errors on path pixels, red dots and line correspond to errors on distractor pixels. Dashed line corresponds to tag propagation error on path pixels for reference. Dotted blue and red lines indicate input-augmented architecture error on path (blue) and distractor pixels (red). Each subpanel corresponds to a different network architecture. From left to right: unrolled recurrent, square mask, neighbors only mask. (C) Performance of the best solution after hyperoptimization for each model type across layers. (d) Results by error type. Models with weight sharing across layers have a constant number of trainable parameters as the number of layers is varied. The circle size indicates increasing layer number. The line connects results for a single model type as the number of layers increases.

Like the input-augmented network, the trained unrolled recurrent network achieved much better performance than the tag propagation solution at low layer numbers, but it quickly plateaued in performance and was surpassed by the tag propagation solution at 15 layers (**Fig. 3B**). The breakdown of error revealed that the network learned a compromise solution with 17% to 25% error on the distractors, but the performance was noticeably worse across all layers than the input-augmented solution. The most likely cause of the increased error was that many solutions learned by the input-augmented network were eliminated through the restriction to shared weights across layers, i.e. the reduction of the parameter space. However, this restriction still did not enable the learning of a successful tag-propagation-like solution. This provides further, though indirect, evidence that the input-augmented architecture learned solutions different from approximate tag-propagation.

The next two architectures imposed further constraints based on the locality of the tag-propagated solution. In the analytical solution (**Fig. 1F**), each neuron performed its calculation based only on its four immediate neighbors, and thus the weight matrix was highly sparse with a pattern based on locality, i.e. which pixels are close to each other in the image. This motivated us to study masking the weight matrix to induce locality-based sparseness (**Fig. 3A**). We denoted the masked weights as $\mathbf{W}^{\mathrm{mask}}$ and $\mathbf{W}_{\mathrm{in}}^{\mathrm{diag}}$, as the recurrent weights will have a sparsity pattern determined by the structure of the image and the input weights will be masked so that each neuron only gets input from its associated pixel value in the input.

We considered two levels of locality restrictions. In the first, we used an $r \times r$ square mask centered on the pixel associated with the hidden units and one weight from the corresponding pixel in the input. Each mask had r^2 parameters and therefore $\mathbf{W}^{\mathrm{mask}}$ has r^2N^2 parameters while $\mathbf{W}_{\mathrm{in}}^{\mathrm{diag}}$ just has N^2 parameters, one for each pixel, for a total of $r^2N^2 + N^2 + N^2 = (2 + r^2)N^2$ parameters. We used $r = 5$, therefore the network had $27N^2$parameters.

Second, we considered a network with a mask that enforces the exact locality structure of the original task such that $\mathbf{W}^{\mathrm{mask}}$ had $4N^2$ parameters corresponding to one weight for each pixel's four nearest neighbors. Each hidden unit also had one weight tying it to the relevant input pixel in $\mathbf{W}_{\mathrm{in}}^{\mathrm{diag}}$ and a bias. Therefore, the total number of parameters is $4N^2 + N^2 + N^2 = 6N^2$.

Note that these masked networks can also be seen as single channel convolutional layers with local connectivity, i.e. without spatial weight sharing such that each pixel learns its own filter. The masked square network learns an $r \times r$ filter for each pixel, and the parameter space can be further restricted by observing that the tag propagation algorithm is spatially invariant, which enables the filter to be shared across all pixels.

We found that at high layer number (25 and 30 layers) the masked networks learned a solution that was as effective as the tag propagation network (**Fig. 3B**). Interestingly, at low layer number (e.g., 10 and 15 layers) the masked networks did not just learn the tag-propagation solution which had lower performance at this number of layers, but rather learned a solution with an error profile similar to successful solutions found at low layer number by the input-augmented networks (**Fig. 3B**). Thus, the masked networks learned an effective interpolation between a compromise solution at low layer number and the tag propagation-like solution at high layer number (**Fig. 3B**). The switch between the solution types occurred around 20 layers when the tag propagation solution becomes more advantageous, and the masked networks learned this solution instead. The 5x5 masked networks were more successful at this interpolation: at low layer numbers, they used the additional weights to learn more complicated and effective solutions, but the enforced locality structure was sufficient for learning to discover a tag propagation-like solution at high layer number.

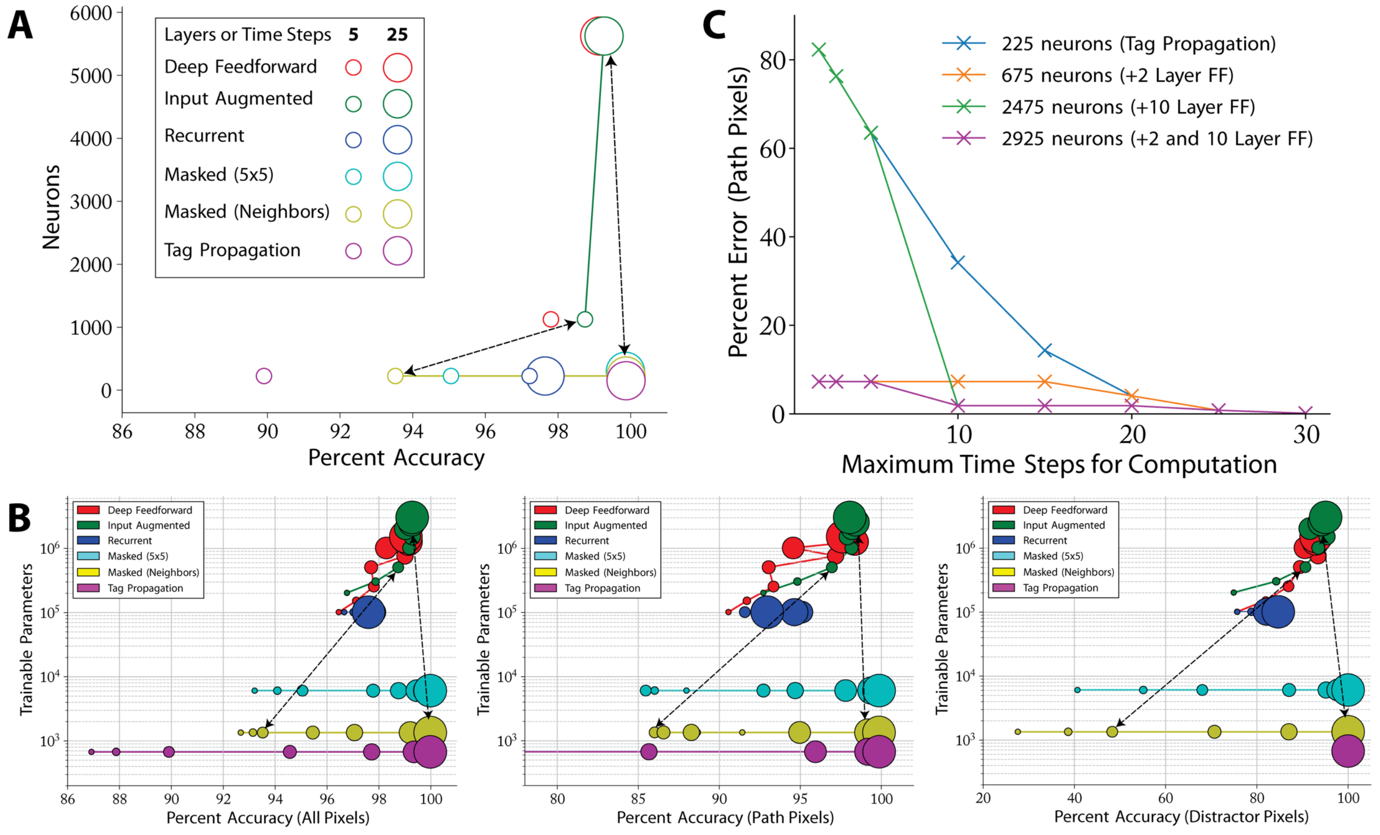


Fig. 4 Performance comparison of all networks on the edge-connected pixel task. A) Performance of the best solution after hyperoptimization for each model type plotted against the number of neurons. Colors correspond to different network architectures. Small circles correspond to recurrent networks run for five timesteps and feedforward networks with five layers. Large circles correspond to architectures that use 25 timesteps or layers. Solid colored lines connect models of the same architecture with different layers or time steps. Dotted lines with arrows highlight the masked and feedforward architectures with equivalent layers or timesteps for ease of comparison. (B) Performance of the best solution after hyperoptimization for each model type across layers with results sub-divided by error type. Models with weight sharing across layers have a constant number of trainable parameters as the number of layers is varied. The circle size indicates increasing layer number. Solid colored lines connect results for a single model type as the number of layers increases. Dotted lines with arrows highlight the masked and feedforward architectures with equivalent layers or timesteps for ease of comparison. (C) Performance of hybrid architecture designed to be able to switch on-the-fly based on how many time steps are available to output a labelling. The model combines the tag propagation solution and trained input-augmented networks; the budget for each curve is the number of neurons the full model requires. The blue curve is the error profile for tag propagation alone while the orange curve shows the result of combining tag propagation with the 2-layer feedforward (abbreviated FF) model. The green curve shows tag propagation combined with the 10-layer feedforward network and the purple curve shows the combination with both the 2-layer and 10-layer networks. Note that the figure only shows path error; the feedforward solutions will also have some error on the distractor pixels.

The architectures described above were composed of different number of neurons and differed in number of parameters by orders of magnitude (**Fig. 4**). When only allowed to run for a few timesteps or equivalently a few feedforward layers, the feedforward solutions outperformed both the tag-propagation solution and the masked solutions. On the other hand when run for a large number of time steps, the tag-propagation and masked architectures outperformed feedforward architectures with layer number equal to the number of timesteps, despite having less than a tenth of the number of neurons (**Fig. 4A**). The advantage of large timestep recurrent solutions was even more striking when considered through the lens of number of parameters. The masked recurrent networks with approximately 10^3 trainable parameters outperformed the fully connected networks with between 10^5 and 10^6 trainable parameters even after extensive hyperoptimization (**Fig. 4B**).

In addition, the masked networks were able to interpolate an effective tradeoff between approximate solutions more powerful than tag-propagation at few layers and solutions similar to tag propagation at many layers that were more powerful than feedforward solutions, yielding high performance across the full range of depths. In fact, these observations can be encoded in a hybrid model which switches between multiple architectures (e.g., one recurrent and one feedforward) according to which is better-suited to the permitted number of time steps on the fly. (**Fig. 4C**). For example, if we compare tag propagation with the trained 2-layer input-augmented network, we see that the feedforward model outperforms tag propagation until 20 time steps. This is because it takes many time steps to propagate the tag along long paths. If we initially set the solution to that provided by the feedforward and only switch to the recurrent timesteps at 20 time steps (if an answer was not required sooner), we achieve a much better error profile across a range to time scales for a small increase in neuron budget, e.g. 7.28% error on path pixels after 5 timesteps vs. 63.52% error from tag propagation alone (orange curve in **Fig. 4C**). Note this model assumes that we optimally switch between the two architectures, i.e., the network is able to switch between the different solutions at the appropriate time step in which a different solution becomes more favorable. This can be further improved by combining multiple feedforward networks but at a higher cost in neurons (the purple curve in **Fig. 4C**). Thus, the hybrid architecture enables one to choose an appropriate trade-off between accuracy at different time scales and the number of neurons used. Notably, this set of results would not have been discovered without the largescale empirical deep learning architecture search.

Taken together, our results demonstrate the importance of recurrent connections and the benefit of knowing task locality or invariance structure in terms of number of neurons which is an important biological limitation, parameter efficiency and trained task performance. We found that learned, spatially masked solutions beyond tag-propagation can be effective whether run for a small or large number of time steps. In addition, these models' local, but not nearest neighbor, connectivity is a better match to neural circuits where neurons are connected with a decaying spatial probability to local neighbors rather than to all other neurons. An alternative solution we explored suggested by our largescale architecture screen overcomes the limitations of recurrent solutions run for few timesteps by using a hybrid architecture in which a short, broadly connected feedforward network (similar to the five-layer input-augmented solution) gives an immediate output when there is a constraint on time but switches to an in parallel developing recurrent architecture solution at a later number of timesteps (e.g., 25), if the dynamics are allowed to propagate that long.

Generalized Tag Propagation and Decision-Making Tasks

In the previous section we used the edge-connected pixel to analyze the differences between feedforward and recurrent computation. Next, we sought to generalize understanding of regimes in which recurrent computation is particularly effective by introducing an extension of tag propagation-like computation and demonstrating how it can be used for decision-making tasks.

While tag propagation in the edge connected task can be viewed as simply generating the correct output by propagating along connected pixels, one can also take a more abstract view which will clarify how this process can be extended. Consider the difference between pixels on the edge and pixels at the center. Determining whether the output for a pixel at the edge should be positive or negative is straightforward, if the input image pixel is positive the output should be positive and if it is negative the output should be negative. In other words, the information required to generate the correct output is fully local. In contrast, for a pixel at the center, whether it is on or off is not sufficient, it depends on different paths from that pixel to the edge, in other words it depends on global information regarding the input. This is precisely why detecting connectivity is a difficult problem. In this viewpoint the goal of tag propagation is to generate an auxiliary variable such that once it is known, computing the output becomes local. Indeed, once tag propagation is complete, the output can be calculated completely locally: if the tag-propagation variable is positive the output should be positive and if the tag-propagation variable is negative the output is negative.

What are the components of this process? First, a certain number of tags will be calculated, in this case just one. Then one starts from setting the tag to positive at a set of seed pixels, in this case the edge pixels. The tag variable can propagate sequentially to neighbors along a given connectivity structure, in this case just a four-connected two-dimensional grid. How propagation occurs given the state of the tag and inputs to a pixel and its neighbors is determined by a propagation rule, in this case if the input to a pixel is positive and one of its neighbors' tag is positive, the tag of that pixel is set to positive. Finally, once propagation completes, the output is computed as a local function of the input to a pixel and the tag, in this case if the tag is positive the output should be positive. Each of these components: (i) number of tags (ii) calculation of propagation seeds (iii) the adjacency structure (iv) propagation rule (v) post-propagation output calculation can be extended, resulting in potentially effective solutions to much more complex tasks. As examples for each of these components: more than one tag variable can be used. The propagation seeds do not need to be constant (i.e., just the edges) but instead could be a function of a particular pattern in the input and would thus change from example to example. The adjacency matrix does not need to be just the regular grid, it could be different matrices. The propagation rule of all tags need not be unique and could even depend on the state of other tags, or multi-dimensional inputs in more complex ways, encoded for instance by a multi-layer Perceptron. Finally, the post-propagation output calculation could have more complex dependencies on the local state of multiple tags and multi-dimensional inputs.

As a concrete example of how such choices generate solutions to more complex tasks and to show the utility of the approach to more biologically-relevant computations we consider a highly abstracted decision making task related to foraging. The task is inspired by decisions that a foraging animal could make upon discovering predators, centrally whether to return to shelter or attempt to hide by freezing in place (**Fig. 5A**). The task has the following setup: one prey animal and several predator animals are placed in a two-dimensional grid environment. The animals can each move only one space left, right, up, or down at each time point. The environment contains a shelter at a specific location; at other locations barriers exist that cannot be passed through. Predators move to catch the animal by moving to the same space it is in. If the animal reaches its shelter first it will be safe. Alternatively, it can choose to freeze. We considered only an immediate decision when the predators are first spotted, whether to run to shelter or freeze in place. In other words, the network must determine whether moving directly to the shelter will be successful (see Methods for full details). As in the edge-connected task, the decision depends on non-local information, e.g., the positions of predators and obstacles.

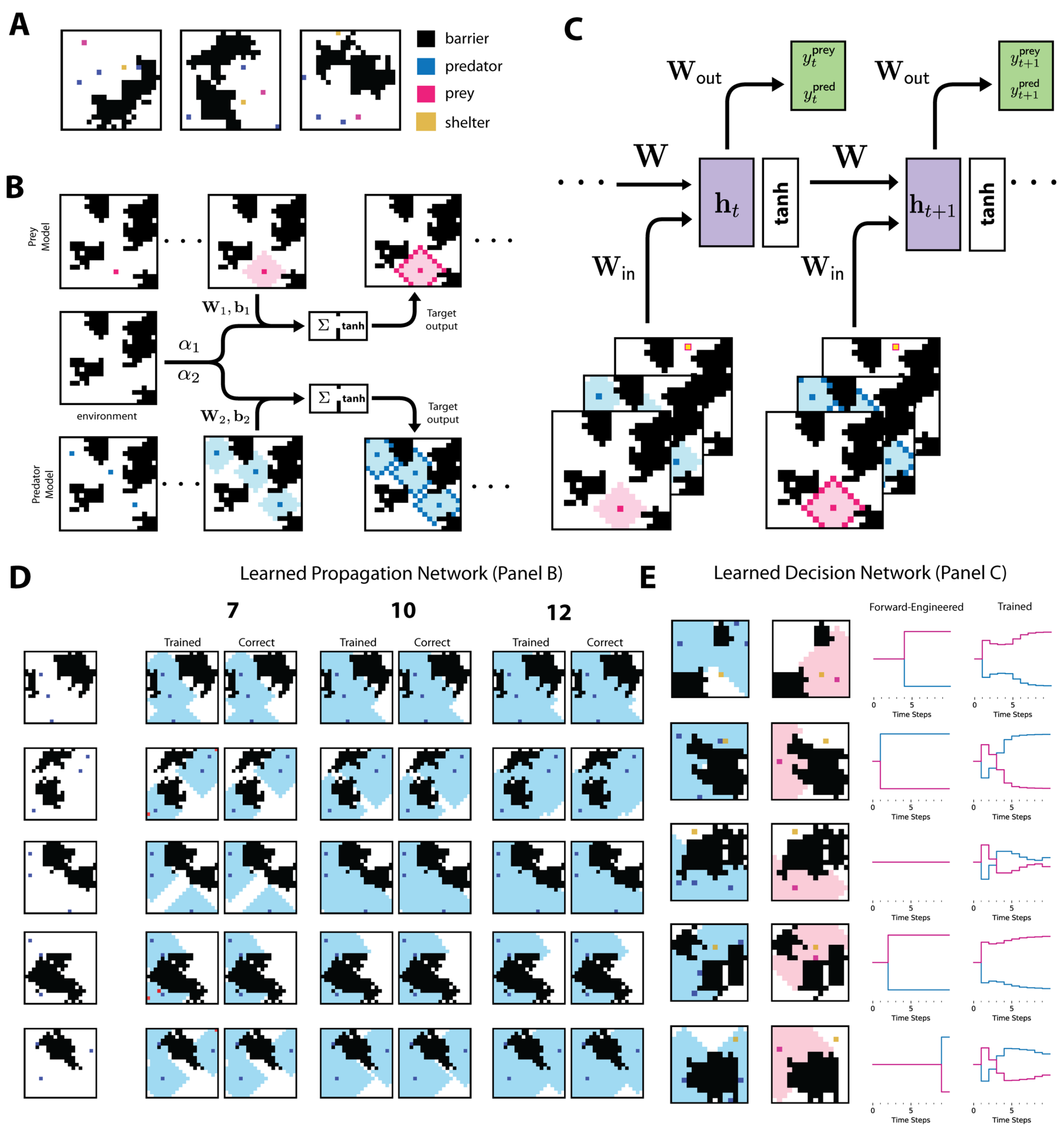


Fig 5. Predator-Prey Task. (a) Randomly generated examples of the predator-prey task. (b) The predator and prey propagation networks. Each one implements the tag-propagation algorithm with open space pixel corresponding to ON pixels and barriers corresponding to OFF pixels. Unlike the edge-connected task, the source pixels which being with the label change every time to correspond to the location of the predator and the prey animals respectively. See methods for further implementation details. (c) Architecture for the trained decision network. The time series of predator and prey ranges are concatenated with the shelter pixel and then are used as the input to a recurrent network. The decision traces shown in panel e are the projection of the trained network onto the two-dimensional readout. (d) Sample outputs for the trained propagation networks. These are the best networks trained with 7, 10, and 12 layers respectively. Errors in the trained network are marked in red. (e) The generalized tag propagation implements a correct version of the decision trace and is described in the Methods. For comparison, we show the decision trace outputted by the trained decision network. Propagation is shown after 10 time steps.

We formalized this task as follows: four $N \times N$ binary inputs were given to a network, each of which was designed to encode information about a spatial environment. Input one was a random environment consisting of several barriers represented by the value -1. The size of the environment was always be the same, and no animal could move onto or through squares that were included in the barrier. Input two was the location of the animal in this environment (the prey location), input three was the location of an arbitrary number of predators, and input four was the location of a safe spot for the animal which we designated the "shelter" (**Fig. 5A**). The task for the network was to appropriately output a "run" or "hide" signal based on whether or not a direct move to the shelter by the prey will be successful. From an algorithmic perspective, the task is equivalent to determining the relative distance between sets of nodes in a random connectivity space (see Methods).

Just as the edge-connected task could be solved efficiently by a network that propagates a tag, the predator prey task can be solved efficiently by a network that propagates multiple, distinct tags. We refer to this solution as generalized tag-propagation. Namely, two distinct tags are propagated by two recurrent networks; one network propagates a tag based on the animal's location, and another network propagates a tag based on the predators' locations (**Fig. 5B**). These networks' output is then combined with an input describing the obstacles and shelter location to generate the architecture's full output, a binary decision to run or hide in place (**Fig. 5C**). Similar to the edge-connected task, solutions for the weights of the propagation networks can be derived analytically and implemented in a recurrent network that then outputs the correct decision labels (see Methods). In essence, this solution learns to generate a tag starting at the location of the animals (predator or prey), propagating out each time to any neighboring pixel that is not a barrier in the environment. In this manner, the hidden state at each time represented all points reachable by the animal or group of predators at time point t. Note that this tag does not represent a specific path taken by the animal, but rather the range of all possible points a path might reach. Given these tags, the output depends on reducing non-local information to a local computation on the state of the tags at the shelter location. We refer to this solution as the generalized tag propagation algorithm.

For simplicity, we trained network architectures separately on the two parts of the proposed solution: (i) generating the two tags and (ii) learning the output from tags and shelter location to the decision. Trained networks were able to successfully perform both parts of the task. First, trained recurrent neural networks accurately learned the propagation (**Fig. 5D**). The propagation networks received input describing the barriers and the initial location of the animal and predators. With these inputs each network was trained by stochastic gradient descent to output the correct shape of the tag at a given number of timesteps (see Methods for details).

Similar to the edge-connected task, we trained the network to produce the correct labeling. Here, though, the state of the propagation at specific times, which reflect the possible paths up until that time point, were of interest (rather than just the last propagated step, as in the edge-connected task). We trained RNNs to run for 7, 10 and 12 timesteps to produce the correct labeling following that number of timesteps (see Methods). Networks successfully learned the propagation: the best 7-layer network achieved 3.50% error; the best 10-layer, 0.0015% error; and the best 12-layer, 0.0009% error (**Fig. 5D**). Trained networks generalized well across number of time steps used: the best 10-layer network achieved 0.0004% error on the 7-step propagation task and 0.0015% error on the 12-step propagation network.

Notably, the trained propagation demonstrates another benefit of the tag propagation algorithm. Learning the predator-prey task requires generalizing the propagation rule to arbitrary starting points. Stated in terms of the edge-connected pixel task, the equivalent of edge pixels which serve as the "source" or seeds of propagation are now wherever the animals are initially located and change from example to example. The tag propagation algorithm has this generalization built in because it gives a correct update rule for any current state of the tags.

The second part of the calculation, distilling the tags and shelter into a decision, is conceptually straightforward: if the tag that corresponds to the prey to arrive at the shelter first via a straight path from the origin to the shelter, then there is no path the predators can take to arrive there first. In other words, if the predator is able to intersect the prey during its run to the shelter then it will also be able to arrive at the shelter first. We trained the recurrent decision network to output the correct choice based on the label propagation given by the solution to the predator and prey tag propagation presented above. Specifically, we used the parameters for the best trained 12-layer propagation network as input to the decision network. The propagated tags were concatenated with the shelter location at each time step and then used as the input to a recurrent network with a 25-dimensional hidden state). Training the network consisted of training the input weights, $\mathbf{W}_{\mathrm{in}}$, the recurrent weights, $\mathbf{W}$, and a set of read-out weights, $\mathbf{W}_{\mathrm{out}}$, which map the hidden state to a two-dimensional vector (see Methods). The loss function for training this network was the binary cross entropy loss between the two-dimensional output vector after 15 steps and a two-dimensional binary label with 1 in the position of the correct decision.

The trained networks successfully learned the task, with the best trained network achieving 90.2% accuracy. To understand the nature of the solution learned by the recurrent neural network, we compared its output over time to the output over time of our analytical solution, the generalized task propagation solution (**Fig. 5E**). As expected, the solutions were not identical, yet comparing the two sets of decision traces showed how the trained network achieved a similar solution by learning a more complex integration of evidence towards the decision compared to the tag-propagated time series over time.

The predator-prey task solution presented here demonstrated 4 of the 5 generalizations described for the predator prey-task: (i) two tags were propagated, one for the predator and one for the prey; (ii) the propagation seeds varied each time based on the initial locations of the animals; (iii) the adjacency structure changed for each instantiation of the task, defined by the location of the barriers; and (v) the post-propagation computation looked at the sequence of tags to determine which reached the shelter first. Note that by incorporating the missing generalization, i.e. (iv) the propagation rule, we can simplify the output computation. Rather than propagating the two types of tags independently, the propagation rule can be modified such that the two tags directly interact and a tag is only propagated to its neighbors if the space has not been claimed by another type of tag (e.g. the predator tag will not propagate to a space already tagged as prey). If run until all locations in the environment are tagged, the output computation to determine which animal is closer to a given space is to simply look at its tag, yielding the run/hide computation for every location simultaneously. An interesting line of future work is to endow each location in the graph with the ability to learn more expressive functions than a single layer of a neural network.

In summary, the predator-prey task demonstrates that extending the tag propagation concept to multiple tag propagation can generalize the tasks for which recurrent networks can turn a global computation into a local one. Moreover, we showed how this type of generalization can be used to extract relevant features for decision making.

Discussion

Here, we performed large-scale computational experiments training a range of architectures on the edge-connected pixel task to analyze the differences between feedforward and recurrent computation. We found in this comparison that the tag propagation algorithm proposed by Roelfsema [17, 18] remained a highly effective solution when run for a sufficient number of time steps and that enforcing the locality structure of this algorithm in the network weights enabled more effective training. Furthermore, a hybrid architecture which switches between feedforward architectures when only given a few time steps and recurrent tag propagation when given sufficient time steps provides superior performance when computational time is sometimes limited and only known on-the-fly. We then introduced a generalized form of tag propagation, using multiple tags in parallel and allowing more complex computations for propagation and showed that propagating multiple tags in parallel facilitates efficient solution of an abstracted decision-making task.

Many previously suggested uses of recurrent and lateral connections can be interpreted in light of generalized tag propagation. Divisive normalization can be thought of as using a tag that calculates global activity levels, contextual interactions such as contour detection are naturally based on specific forms of tags (e.g., tags that trace contours) and predictive coding can be viewed as shaping the locality structure and propagation function by the average statistics of the environment. Similarly, the temporal use of recurrence, common in machine learning, can be seen as transforming a task that requires a global computation across time to one that can be performed locally on any time point by selectively learning and propagating a tag through time, just as the tag propagation we discuss transforms a global computation of connectivity to a local one given the input and a tag. Indeed, the internal states of RNNs performing a task can be seen as a combination of highly entangled propagated tags. While this perspective does not necessarily improve our understanding of these well-established uses of recurrence, it suggests additional possible applications for more complex tag propagation and more interpretable architectures.

Machine learning attention-based networks and hierarchical computation

Recent advances in machine learning have achieved state-of-the art results by replacing typical RNNs and their associated hidden states with “attention”-based networks. Such networks do not have large hidden states that evolve continuously over time, but rather include a component, dubbed an attention mechanism, that learns what part of the large sequence of inputs to focus on when computing input transformations. Examples of such networks include Transformers for NLP [25] and Graph Attention Networks for graph-structured data [26]. Moreover, variants of such architecture use hierarchical structures of attention to better match the hierarchical structure of data and discover the context in which particular patterns are informative rather than simply filtering particular patterns [27]. Similarly, previous work has suggested recurrent connections are important in hierarchical frameworks, including Brosch, Neumann, and Roelfsema in the context of contour tracing [21] and Jehee, Lamme, and Roelfsema in the context of boundary assignment [20]. In these papers, the authors assume the network uses a hierarchical approach to extract features in the image, i.e. that deeper layers contain representations of coarser features that are made up of higher-resolution features in the layers below. In this manner, the network can iteratively extract large-scale information about the image to make global decisions about whether points are connected. Recurrent connections then serve the important purpose of communicating information about coarser features to lower levels, perhaps to shift computation to parts of the image more relevant for decision making. These explanations match nicely with the success of attention-based hierarchical models. In a loose sense, the multi-tag propagation approach for the predator-prey task has a similar flavor, introducing intermediate variables that modulate how inputs are used by the decision layer. Though we focused on parallel, non-hierarchical tag propagation, one can have tags propagate and interact across multiple levels, leading to hierarchical versions of tag propagation. Along these lines, Nayebi et al. [28] perform a model architecture search over CNNs with local and long-range recurrent feedback to obtain improved performance on ImageNet.

More generally, ideas related to our results on a generalized framework of identifying tasks suited for recurrence have been put forward by Poggio et al. in [29]. Their main aim is to understand how the number of parameters required to achieve an ϵ-approximation of the function scales in deep and shallow networks. They point out that deep networks are much better suited to functions that are a "hierarchical composition of local functions" compared to shallow networks, where local refers to bounding the number of inputs from the previous layer to each neuron. Our concept of local functions is similar except that we assume the locality and function structure are shared across the compositions, and but to the best of our knowledge no one has shown the full generality of this framework. Finally, other work on visual processing has considered the use of local connections in parts of gated recurrent architectures to enhance their performance on contour tracing tasks [30].

Benefits of Generalized Tag Propagation

The tag propagation networks trained very efficiently as we were able to greatly reduce the number of parameters to be learned by three assumptions. First, we masked weights according to a locality structure. Second, we assumed that the propagation function was identical across most units and shared these weights. Third, we assumed that the propagation function was identical across time, yielding a recurrent solution. Naturally, these assumptions restrict the class of tasks that can be solved in this fashion. In the generalized tag propagation framework, however, we demonstrated that the class of tasks can be substantially expanded by increasing the number of such networks, allowing for different adjacency structures for propagation, and making their constituent computations more complex. Learning the task can thus be sub-divided into three processes that result in a transformation of global information into local features. First, learning the adjacency structure that defines how information propagated. Second, learning the propagation function on top of this adjacency structure. Third, learning output-like functions atop the tag-propagation results. We hypothesize that given the common presence of fan-out architectures, e.g., the large ratio of cortical neurons to the number of input neurons from thalamus, the number of parallel tags that can be propagated may be quite large. As the number of tags expand, the distinction between such networks and a generic RNN begin to blur. Nonetheless, we believe that generalized tag-propagation dynamics are highly interpretable, and thus beyond the applicability to neural circuits, such solutions can be thought of as imposing a bias on a network to help with interpretability of the computation.

While generalized tag propagation relaxes the assumptions on an underlying connectivity structure by expanding the expressibility of the architecture, the presence of such a structure is a key assumption. Importantly, many computational tasks have a structure to them that can be viewed as an underlying adjacency structure, even if it is more abstract. For instance, in reinforcement learning, such structure might be the graph of accessible next states defined by the transition matrix, and in inference, such structure might be the probabilistic graphical model of the distribution. However, in many real-world tasks the locality structure is unknown. Mechanisms to infer tag propagation strategies in such contexts are an important direction for future research. Interestingly, our results in the edge-connected pixel task suggest that even an imperfect understanding of the structure, as is likely be to be inferred by learning or coarse-grained guesses such as the masked 5x5 square networks we studied, already show computational advantages. Moreover, as pointed out by Kushnir and Fusi, even without specific task structure, propagation in recurrent neural networks allows information to spread, resulting in locally-connected networks that have access to all necessary information and still perform global computations [31].

Empirical Experiments in Deep Learning

Deep learning based solutions have been remarkably successful at a range of computational tasks [22, 23]. Part of their appeal is their wide applicability, the ability to transform a set of architectural constraints and samples of input-output pairs into a concrete solution to a difficult computational problem. At the same time there is no guarantee that stochastic gradient descent converges to the global optimum, and therefore one cannot be certain that solutions found by training neural networks truly represent the best solution possible by a given type of network or certain architecture class. For these reasons, multiple paradigms and approaches for using large-scale empirical experiments are a subject of substantial recent research. [32-35] Here, we have used two tools to ameliorate this issue: per-architecture hyperoptimization and training multiple models from random initializations. In our experiments, we performed independent hyperoptimization for every layer and model combination, which required approximately as much computational time as the experimental runs themselves. Specifically, for each network architecture and layer combination, 100 models were trained for the hyperoptimization followed by 50 training runs from random initializations. This process ensured that the training parameters were not tuned to any particular depth of network and allowed us to be reasonably confident that we were observing how the trained solution evolved with increasing layer number without other confounding factors. Despite the complexity of our tasks we were able to show consistent and logical changes across the architectures we trained and, in combination with training masked models, our findings suggest we were able to minimize issues of sub-optimal training. These tools, however, do not fully solve the general problem of neural network optimization.

How best to leverage the ability of deep learning to find empirical solutions to tasks in the presence of imperfect training remains an open question, and understanding the algorithmic utility of recurrent connections is certain to make substantial contributions to the discussion. Understanding the algorithmic utility of recurrent connections is an ambitious scientific question—with many possible answers—whose study will require sustained research inquiry. Nevertheless, we consider our work as a meaningful step towards this understanding. The large-scale deep learning experiments we report here robustly demonstrate how recurrent solutions can be more efficient. They also reveal interpolations between classical solutions to the connectivity problem and solutions that are effective in a fast, few-timestep computational regime. In addition, our generalization of multiple tag propagation and identification of the structure of tasks that benefit from recurrent computation further the understanding of why recurrent and lateral connections might be so ubiquitous in neural circuits.

Methods

Code

The code for running the experiments in this paper and the associated data produced is available in the following Github repositories:

- Edge-connected pixel task https://github.com/druckmann-lab/edgeConnectedPixel
- Predator-prey task https://github.com/druckmann-lab/predPreyTask

Previous Networks Architectures for Detecting Connectedness

In this section, we detail the mathematical form of the networks discussed in the introduction for detecting connectedness. In Minsky and Papert's definition of a perceptron in [16], the scalar output y is assumed to be a function of d-predicates $\psi_1(\mathbf{I}), \dots, \psi_d(\mathbf{I})$, i.e. d functions of the input image $\mathbf{I}$. The perceptron function is then defined by a d-dimensional vector $\mathbf{w}$ and threshold b:

$$y = \begin{cases} 1 \quad \text{if} \quad \left(\sum_{i=1}^{d} w_i \, \psi_i(\mathbf{I}) \right) + \mathrm{b} > 0 \\ 0 \quad \text{otherwise} \end{cases} \tag{6}$$

If the predicates ψ_i are allowed to be arbitrary non-linear functions, then this network is a support vector machine combined the kernel trick. The non-linear functions form a new feature space in which we learn a hyperplane boundary for classification. In practice, the predicates are defined by the user and the weights and biases are learned.

Minsky and Papert's insight was to consider how the size of the perceptron had to scale with the image size in order to perform the connectivity task when the predicates were restricted to be "local." In their definition, a predicate is local if the number of image pixels used to calculate each predicate is bounded (the bound is referred to as the order of the predicate). Minsky and Papert demonstrated that there was no set of finite order predicates which could perform the binary connectivity task as the size of the image was scaled, concluding that connectivity is a fundamentally serial task. [16] Intuitively the problem is that the connectivity between two pixels can altered by flipping any bit in the image meaning the perceptron can only perform the task if the predicates have access to all the pixels in the image.

If we limit the predicates ψ_i to be linear functions followed by a shared non-linearity, then the perceptron considered by Minsky and Papert is equivalent to the modern usage of a perceptron with one hidden layer. It first computes a hidden vector $\mathbf{h}$ as a function of the input $\mathbf{x}$ according to the function $\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$, where $\mathbf{W}_1$ is a matrix and σ a non-linear element-wise function. The output $\mathbf{y}$ is then computed in terms of this hidden layer: $\mathbf{y} = \sigma(\mathbf{w}_2^{\mathrm{T}}\mathbf{h} + \mathbf{b}_2)$. The weights and biases of both layers are typically learned via backpropagation. As discussed by Roelfsema [17, 18], single hidden layer feedforward networks can solve the two-pixel connectedness task using a hidden unit for every possible path between the two pixels and the output layer returns +1 if one of these hidden units is activated and -1 otherwise. However, in order for this solution to perform the task perfectly, the number of hidden neurons must equal the number of patterns without loops that connect the two pixels and thus scales exponentially with the size of the image. (Note that if we want to detect all pixels connected to a set of points such as the edge pixels, some of the patterns can be shared but the scaling will remain exponential in the number of pixels.) Such a solution illustrates Minsky and Papert's concept of non-local computation as the longest pattern scales in length with the size of the image and thus a hidden unit potentially needs access to the whole image [16]. Concretely, consider the central pixel of the image. This pixel can be connected to the edge by a non-overlapping path that forms a spiral around this pixel. Detecting this pattern requires input from pixels throughout the image, each of which can toggle its value and change the output.

As discussed in the Introduction, a much more efficient solution is tag propagation via a recurrent architecture. This algorithm has a number of advantages over the single hidden layer perceptron. First, the number of calculations at each time step scales on the order of the number of pixels and can these computations can all be performed in parallel. Second, the calculations are highly local: they require the value of the pixel and its four neighbors, a property which does not change with the size of the image. Thus, by using a recurrent vs. feedforward architecture we are able to realize Minsky and Papert's ideal of neural computing: many repeated local computations performed in parallel solve a task that is global in nature (it requires all the pixels in order to make a decision).

This solution, however, has an additional advantage that becomes evident when we consider the finding all pixels connected to the edge of the image, which we call the edge-connected pixel task. Here we are interested in finding all pixels connected a set of source points (the edge pixels) rather than a single point. Unlike the pattern matching solution which requires a significant increase in hidden units to account for the additional source pixels, the tag propagation algorithm can remain the same and perform the computations in parallel. We simply initialize the initial state to have a connected tag for all the source pixel and the recurrent network will propagate the tag in parallel, all via local computations.

In [17, 18], Roelfsema considered a pyramidal modification to the tag propagation to reduce the number of time steps required on average to converge to the correct solution. In the standard tag propagation algorithm, the network must run time steps equal to the length of the longest path in the image for which no shorter path exists. In the worst case, this scales with the number of pixels in the image (e.g. the non-overlapping spiral from the edge to the center pixel). In the pyramidal algorithm, the first level of the hierarchy performs the tag propagation in the normal manner. In the second layer, each pixel is connected to a 3x3 square of 9 pixels in the first layer and activates only if this square has no disconnected components. In this layer the same tag propagation rules apply. This is repeated recursively until the top layer has only 1 pixel, and thus the network forms a pyramid of recurrent networks. If the pattern is such that each 3x3 grid of the image has no disconnected components, propagation can occur much more quickly through the higher layers of the pyramidal architecture. Note that this does not decrease the number of trainable weights, but rather increases the speed of propagation.

Edge-Connected Pixel: Experimental Set-up

Here we discuss our experimental design choices which seek to mitigate confounding factors in comparing the inherent capabilities of a given architecture to perform the edge-connected pixel task. These include the design of the sample generator, the training and hyperoptimization procedure, and the addition of skip connects to certain deep feedforward models.

For the experiments in this paper, we generated samples with a 15x15 image size (**Fig S1**). Each pixel in the image is encoded to be off if it has a value of -1 and on if it has a value of +1. The sample generator starts with an image of all -1's and then randomly turns a set of seed pixels to +1 by independently drawing an independent Bernoulli random variable with success probability 0.05 for each pixel. For each of these seed pixels, we perform a random walk in the following manner: continue straight with probability 0.65, turn right or left with probability 0.125 each, or terminate the path with probability 0.1. The initial direction of the walk is chosen uniformly over the four cardinal directions and each path is continues the walk until it terminates or hits the edge of the image.

We next seek to increase the number of distractors by randomly disconnecting paths from the edge. This is done rather than randomly turning on pixels in the image because it creates distractors with clusters of pixels that look more like paths. The procedure is defined by a cutoff and a disconnect probability. The cutoff determines how far along the path to disconnect, and the disconnect probability gives the independent success probability for disconnecting any path. For our experiments we uniformly chose the cutoff to be 0 or 1, meaning disconnecting the path meant turning OFF either the pixel on the edge of the image or one step away from the edge. The sample generator then finds all such pixels and turns them off with 80% probability.

The final step of the sample generator is to ensure a user-defined distribution over path lengths. For any connected pixel in the image, we can find the shortest path to the edge via on pixels. A simple measure of the complexity of a task sample is the largest such value over all pixels, i.e. what is the longest path from the edge to some pixel for which there are no other shorter paths. This property is easily calculated using the tag propagation solution: start with a connected tag for all edge pixels which are on and count the number of propagation steps until the set of labelled pixels stops changing.

To enforce the requested distribution, the sample generator first calculates based on the total number of samples requested how many samples of each path length it needs to achieve this distribution. For our experiments, we used a uniform distribution over path lengths from 1 to 25 so that if n samples were requested, the generator would need n/25 samples for each path length in this range. Each time a sample is generated by the random walk method, we calculate its path length and then check how many of this path length we already have. If we already have n/25, this sample is thrown out; otherwise, we keep it and increment the count for this path length. If a path length is over 25, it is counted in the quota for path length 25.

The sample generator guarantees the path distribution requested by the user, but it does not guarantee a certain number of path or distractor pixels. Generating 100,000 samples we observe the following empirical distribution over pixel types: 15.9% path pixels, 8.3% distractor pixels, and 75.8% off pixels. This achieves our goal of forcing both of the trivial solutions to the task to have high error. Labelling all the pixels as "disconnected" means an overall error of 15.9% while labelling all the on pixels as connected gives an error of 8.3% as illustrated in **Figure 2D**.

Next, we consider network hyperoptimization and training. The training loss function ℓ is a function of both the labelled image and the output of the network and determines what the network optimizes for during training. We use the mean squared error (MSE) between the full image outputted by the network and the correctly labelled image from our sample generation: $\ell(\mathbf{y}, \mathbf{y}^*) = \|\mathbf{y} - \mathbf{y}^*\|_2^2$ or the squared Euclidean distance. This loss treats performance on all pixels equally and rewards the network for outputting labels closer to the true label +1 or -1. Note that this loss function treats the task like a regression problem, but when analyzing the output, we want to evaluate the classification accuracy of the pixel. This is done in our experiments using the sign of the network output: a positive output means the pixel is classified as +1/connected and a negative output correspond to a classification as -1/disconnected. Alternatively, one could use the binary cross entropy loss on each pixel for training, but this is not done in this paper.

All networks were implemented in Pytorch [36] and trained via its implementation of ADAM [37], a stochastic first-order optimization algorithm like stochastic gradient descent (SGD). In order to compensate for the effects of the choice of training parameters, we performed hyperparameter optimization over three parameters of ADAM for every architecture and layer combination: the learning rate, the batch size, and the weight decay. The learning rate is related to the size of steps the algorithm takes through the parameter space, the batch size is how many training examples are used in each estimate of the gradient, and the weight decay is a regularization parameter which penalizes to the ℓ_2 norm of all the weights in the model.

The most effective method to perform hyperoptimization is still an open research question in machine learning. We used a hyperband, a bandit hyperoptimization algorithm that begins with a large number of randomly chosen hyperparameters sets and then trains a network corresponding to each set of sampled parameters. [38] The first step is to select hyperparameter triples uniformly across a given range or equivalently selecting random points in a three-dimensional space. For learning rate and weight decay, the hyperparameters we selected uniformly over the logarithmic space, i.e. the exponent of the parameter was sampled uniformly over the range -2.5 to -5 rather than uniformly over the raw value $10^{-2.5}$ to 10^{-5}. The batch size was sampled uniformly between 32 and 256.

For each sampled triple, we randomly initialize a network and set up an optimizer with these parameters. In random search hyperoptimization, all of these networks would be trained for a given number of epochs and then the best performing network would be selected. The insight in the hyperband algorithm is to instead break this full training run into hyperband epochs after which the models are pruned based on network performance. In our experiments we started out with 100 models each and used a hyperband epoch of 1000. A given run would train the 100 models for 1000 epochs, evaluate their performance, and then eliminate the 50 models that performed the worst. The remaining models are trained for 1000 more epochs after which the networks in the bottom half of performance are again eliminated. This process is repeated until 5000 total epochs are reached, and the algorithm returns the parameters of the best performing model.

The main advantage of hyperband over random search is computational complexity; networks in the bottom percentiles of performance are eliminated early, speeding up the subsequent epochs. Hyperoptimization, however, was not performed for the masked models, i.e. the square 5x5 mask and the neighbors mask. The first reason is that weight decay should be set to 0 for these models as the sparsity pattern in the weights has already been enforced by masking. In the fully connected model, weight decay pushes the weights towards 0 which makes sense as a regularization technique when you assume the computation has locality structure. However, when this locality structure is already encoded in the weights, the regularization can be counter-productive. Second, not performing hyperoptimization emphasizes how knowledge of the locality structure makes learning easier; it sets the masked models at a disadvantage, but we see in experiments that they still easily learn the efficient tag propagation algorithm.

Lastly, we consider the modifications made to deep feedforward networks. A common problem in such networks is the vanishing gradients in which the gradients become vanishingly small as the algorithm backpropagates through many layers [39, 40]. In our experiments, once the deep feedforward network had more than 15 layers, it got stuck across many initializations in the trivial solution that labelled all pixels as OFF. A common solution first proposed for image processing is the addition of skip connects to the network, or direct connections from early layers to later layers in the network [41]. We thus added a skip connect with fully connected weights to the input every four layers for the deep feedforward networks with 15, 20, 25, and 30 layers. Note that these skip connects take the same form as the input augmentation, and if we added a skip connect to every single layer, the two networks would be identical. Intuitively, the skip connect allows the network to reference the input image every four steps rather than every step as in the input augmented network.

Predator-Prey Task: Experimental Set-up

To generate samples of the predator-prey task we begin by creating a random environment (**Fig S2A**). We used 20x20 images for which -1 indicates that a pixel is part of the barrier in the environment and +1 indicates that it is open space. In the framework of the edge-connected task setup, this means that barriers are off pixels and open space are on pixels. We begin with no barriers, meaning all pixels are set to +1, before laying down between two and five rectangular barriers, the number being chosen uniformly in this range. For each rectangle, the location of the upper left corner is chosen uniformly over the pixels of the image (excluding the last three columns and rows to ensure most of the rectangle is in the image) and the width and height are chosen uniformly between 2 and 8. This procedure can result in the rectangle stretching beyond the left or bottom side of the image in which case the rectangle is cut off.

The environment now consists of several rectangular barriers which we then randomly expanded. For every pixel on the edge of a rectangle, we drew eight independent Bernoulli random variables with success probability 0.15. Each one of these corresponds to one of the eight directions one can travel from the pixel, including the four diagonal directions. For every direction that succeeds, if the pixel was not already part of the barrier we changed its value and added it to the queue to perform the same random expansion procedure. We continued this process until the queue was empty at which point the environment was finalized. Lastly, we assigned locations to the predator, prey, and shelter, each of which was chosen uniformly among the open pixels. We used one prey animal, three predator animals, and one shelter location.

We next turn to the details of implementing the generalized tag propagation solution to the task. Our approach was to decompose the problems into two sub-computations: (1) generating the multi-tag propagation and (2) making a decision based on the multi-tag propagation. We will show that the multi-tag propagation, as its name suggests, can be generated by a generalization of the edge-connected task and thus can be implemented via a recurrent neural network. We then designed a decision network that has access to the sequence of representations produced in a streaming fashion (i.e. one at a time, not all at once).

Consider first the prey animal. At every timestep t, we can define an $N \times N$ matrix R_t which specifies all points in the enviornment that can be reached by the animal, accounting for all possible paths. The elements of R_t have a one-to-one correspondence to the positions in the environment and we will use the labelling +1 for reachable at time t and -1 for unreachable at time t. Importantly, the range does not consider a single path, but rather all paths simultaneously.

An instructive example is to consider how the range propagates in the case of an environment with no barriers. R_0 will have a single reachable point: the starting point of the prey. Any point adjacent to this starting point will then be reachable in the next step, so R_1 will label both the starting point and its four nearest neighbors. To compute R_t from R_{t-1}, we simply need to look at each reachable point in R_{t-1} and label all of its neighbors as reachable. In this manner, for a two-dimensional grid environment with no barriers, the range will propagate out in a diamond-shaped pattern until it reached the edge of the environment. For sufficiently large t, the range will be the entire environment.

Now consider the modification where we add random barriers. The procedure for propagating out the range will remain the same except if a reachable point has a barrier as a neighbor, it will not be added to the range in the next iteration. The range will propagate out in the same manner, except it will stop whenever it reaches a barrier. Furthermore, the barriers may divide the grid into disconnected components where there is no way around the barrier to a certain set of points. If the environment is connected, R_t for sufficiently large t will be all non-barrier points in the environment; otherwise for disconnected environments, R_t will converge to the set of non-barrier points in the region in which the prey starts.

We can define the same sequence of hidden representations for the predator animals, but now instead of keeping track of the paths of one animal, we need to keep track of the paths of an arbitrary number of animals. This is simplified, however, by the fact that in our task definition we do not care which predator is able to overtake the prey. We can then track the range R_t as the set of all points reachable by any of the predators. R_0 will be the initial location of the predator animals and at each propagation in time we turn on all pixels that neighbor a reachable pixel and are not a barrier pixel. Importantly, the complexity of calculating the range does not change with the number of predator animals; at each step, all the pixels need to check whether any of their neighbors are reachable and change their label to reachable unless they are a barrier pixel.

To implement this procedure analytically, we first observe that calculating the range for either the predator or the prey can be viewed as a modification of the edge-connected pixel task. The environment is a set of on/off pixels, with the barriers as the off pixels. Instead of starting our tag propagation from the edges of the environment, we start from a small number of ``seed'' pixels that correspond either to the set of prey or predator animal starting points. We then propagate out a ``reachable'' label from these points to neighboring on pixels. The important difference now is we are not interested in the final labelling, but rather the sequence of labellings which tells us when a given point becomes reachable. Providing the predator/prey animal locations as the initial conditions, the range propagation can be implemented by an identical recurrent network of the tag propagation algorithm as illustrated in **Figure 5b**.

From the computational point of view, calculating hidden representations in this manner also has a useful parallelism. The network can easily calculate the two ranges (one for the prey animal and the other for the predator animal) at the same time since no interaction is required between the two computations. In fact, an arbitrary number of representations, each perhaps representing useful information for a decision network, can be calculated in parallel provided there are not interactions between the computations. This is the same benefit of multi-tag propagation observed for the generalized task framework.

We now turn to the decision network that receives as input the sequences generated by the multi-tag propagation, i.e. at each time step it receives the two ranges $\{\mathbf{R}_t^{\mathrm{prey}},\ \mathbf{R}_t^{\mathrm{pred}}\}$ and the location of the shelter pixel. In the main text, we simply concatenated these inputs together and trained an RNN on the task (**Fig. 5C**). The RNN has a 25-dimensional hidden state, used the tanh non-linearity, and learned read-out weights to a two-dimensional output. Passing this output through the softmax operator gave probabilistic belief of the network that the prey was closer to the shelter (the first dimension) or the predator was closer (the second dimension). Training was done via the binary cross entropy loss.

For comparison, we also forward engineered a recurrent network to implement what we define as the ground truth solution for the problem (**Fig. S2B**). Consider what this input looks like for a single pixel in the environment that is not the starting location of any animal. At some time point in the prey's range, the labelling may switch from unreachable to reachable and then will remain reachable for the remainder of the time steps. The time point at which this switch occurs indicates the length of the shortest path from the predator that reaches this pixel given. If the point remains unreachable up until time T, there are two possible causes: (1) The length of the shortest path to this point is greater than T or (2) the point is disconnected from the prey in the environment in which case no path exists. In the same manner, the length of the shortest path can be extracted from the predator range.

Once we have the propagated tags, determining whether the prey or predator can reach any given point first is straightforward: whichever group has the tag which arrived earlier can reach the point first. If we perform this calculation for the shelter pixel, the prey should go for the shelter if its tag arrives first at the shelter and hide otherwise. This calculation could be performed for an arbitrary number of pixels given the same hidden representation. When specifically considering one pixel (in this case the shelter pixel), the network should give no preference to the predator or prey until one of the ranges reaches this pixel; afterwards the probabilistic belief should switch to be 1 for this group (and 0 for the other) because the network has established that this group is closer.

This decision layer can be implemented correctly using a two-neuron recurrent neural network with strong inhibitory weights (**Fig. S2B**). Denoting these two neurons as $\mathbf{y}$, we defined the relation between decisions and $\mathbf{y}$ in the following manner: $\mathbf{y} = [1, 0]$ indicates the prey should go for the shelter because it is closest to it and $\mathbf{y} = [0, 1]$ indicates it should hide because one of the predators is closer to the shelter. Denote by $\mathbf{C}$ an $N \times N$ matrix of zeros except for the location of the shelter which is denoted by a 1 (a simple conversion from the original shelter input which used +1/-1 to indicate the shelter location). At each time step, $\sum\left(\mathbf{R}_t^{\text{prey}} * \mathbf{C}\right)$ will be 1 if the shelter is reachable for the prey and -1 otherwise, given that $*$ denotes the element-wise product of two matrices and the sum is over all the elements of the results matrix. The equivalent signal for the predator can be extracted in the same manner: $\sum\left(\mathbf{R}_t^{\text{pred}} * \mathbf{C}\right)$.

Lastly, we need to add inhibition between the two neurons. We only want the neuron for either the predator or prey to turn on thus indicating which group was able to reach the shelter first; otherwise our network will output $\mathbf{y} = [1,1]$ in the common case where the shelter is eventually reachable by both groups of animals in time T, giving no indication which group was closer. The recurrent dynamics of the decision network are then given as follows:

$$\begin{pmatrix} y_t^{\text{prey}} \\ y_t^{\text{pred}} \end{pmatrix} = \sigma\begin{pmatrix} w_1 \cdot \sum\left(\mathbf{R}_t^{\text{prey}} * \mathbf{C}\right) - w_1^{\text{inhibit}} \cdot y_t^{\text{pred}} \\ w_2 \cdot \sum\left(\mathbf{R}_t^{\text{pred}} * \mathbf{C}\right) - w_2^{\text{inhibit}} \cdot y_t^{\text{prey}} \end{pmatrix} \tag{7}$$

The decision network thus has four learnable parameters: w_1, w_2 and $w_1^{\text{inhibit}}, w_2^{\text{inhibit}}$. Under the following circumstances, the network will output a correct solution to the predatory prey task: (1) there is no sample where the shelter is not reachable by either the predator or the prey, (2) for every sample, the number of time steps the network is run T is larger than the shortest path to the shelter by any animal, (3) all the weights are positive values large enough to make the slope of the sigmoid steep (a value of 10 or greater is sufficient), and (4) for each neuron the inhibitory weights are larger than the weight on the network output.

Acknowledgments

B.W.L. was supported by the Department of Energy Computational Science Graduate Fellowship program (DE-FG02-97ER25308). S.D. is supported by NIH grant R01EB028171 and Simons collaboration on the global brain grant 542969SPI. We would like to thank Byungwoo Kang, Minseung Choi, Winfried Denk, Jonathan Amazon and Alla Karpova for discussions and feedback on earlier drafts of this manuscript.

Supporting Information

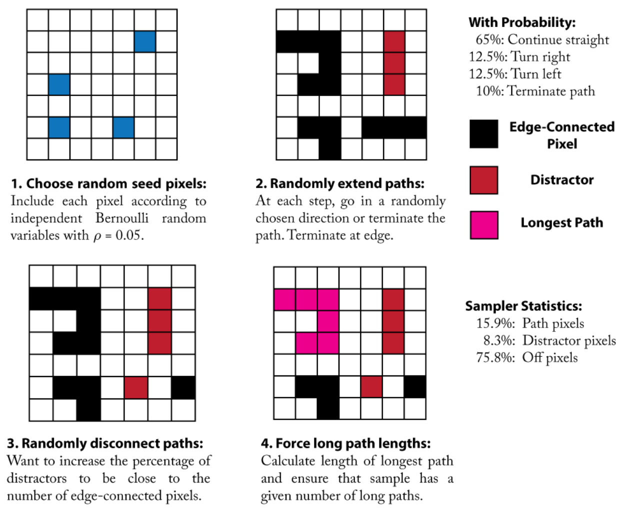

With Probability:
65%: Continue straight
12.5%: Turn right
12.5%: Turn left
10%: Terminate path
Edge-Connected Pixel
Distractor
Longest Path
1. Choose random seed pixels:
Include each pixel according to independent Bernoulli random variables with ρ = 0.05.
2. Randomly extend paths:
At each step, go in a randomly chosen direction or terminate the path. Terminate at edge.
3. Randomly disconnect paths:
Want to increase the percentage of distractors to be close to the number of edge-connected pixels.
4. Force long path lengths:
Calculate length of longest path and ensure that sample has a given number of long paths.
Sampler Statistics:
15.9%: Path pixels
8.3%: Distractor pixels
75.8%: Off pixels

Fig S1. Sample generation procedure for the edge-connected pixel task.

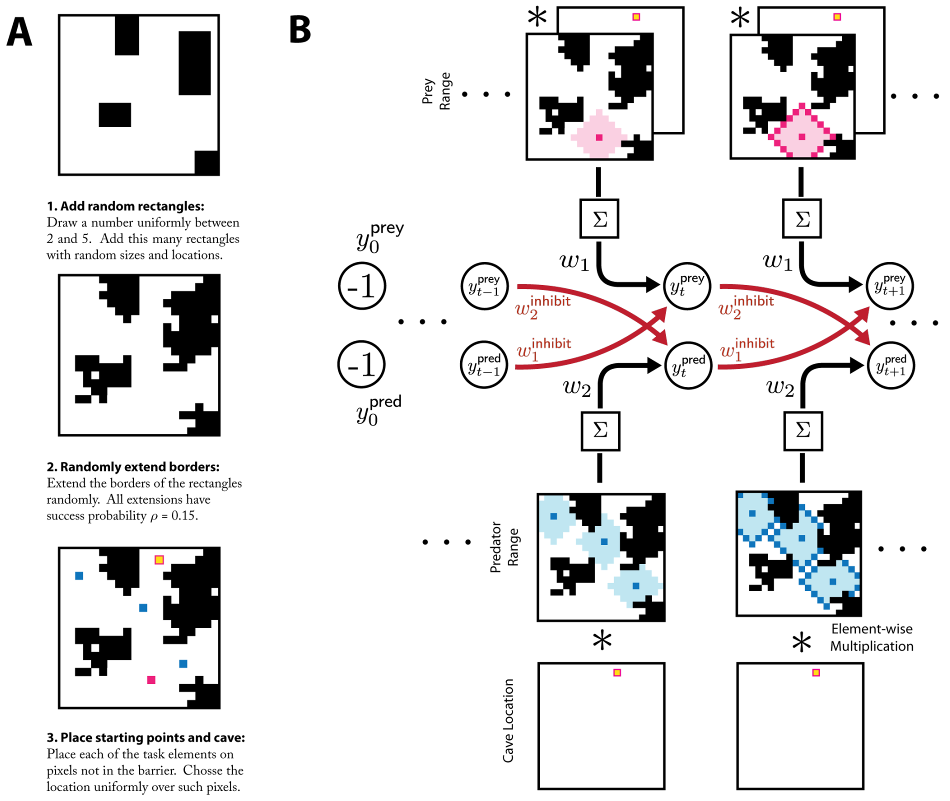

A
1. Add random rectangles:
Draw a number uniformly between 2 and 5. Add this many rectangles with random sizes and locations.
2. Randomly extend borders:
Extend the borders of the rectangles randomly. All extensions have success probability ρ = 0.15.
3. Place starting points and cave:
Place each of the task elements on pixels not in the barrier. Chosse the location uniformly over such pixels.
B
Prey Range
Predator Range
Cave Location
Element-wise Multiplication
Σ
w1
w2
w2 inhibit
w1 inhibit
-1
y0 prey
y0 pred
y t-1 prey
y t-1 pred
y t prey
y t pred
y t+1 prey
y t+1 pred

Fig S2. Experimental details for predator-prey task. *(a) Sample generation procedure for the predator prey task. (b) The recurrent decision network composed without output of the propagation network in the generalized tag propagation algorithm. At each time step the network extracts whether or not the shelter pixel is the in range of either group of animals. Once it comes into range for one group, the corresponding neuron activates and inhibits the other neuron. The active neuron in the final time step indicates which group was closer to the shelter.*

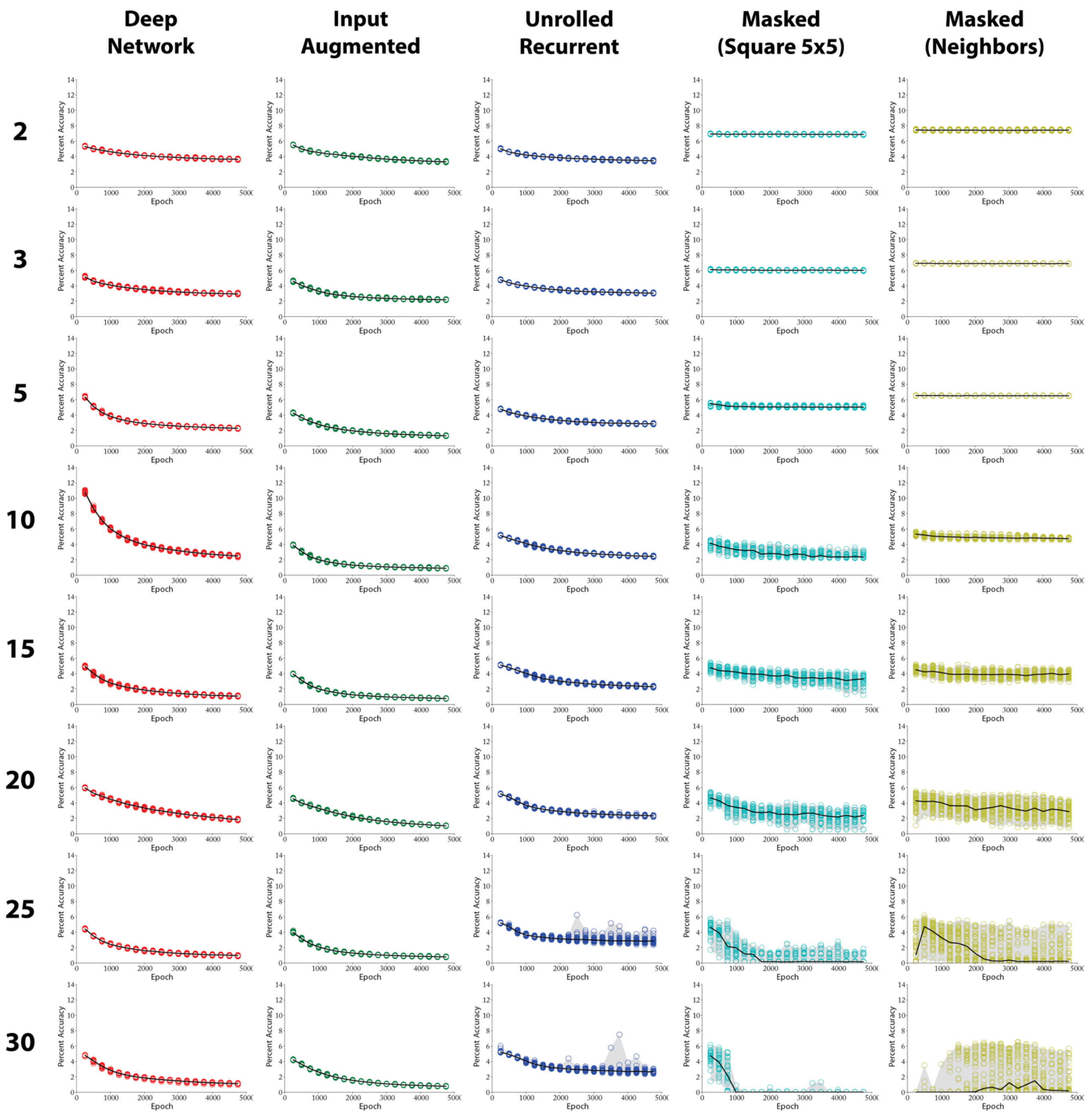


Fig S3. **Total Error on All Runs.** For each layer/architecture combination, 50 models were trained from random initializations for 50 epochs following hyperoptimization. Here we plot the error on all pixels every 250 epochs. The gray shading indicates the range of accuracies across all instantiations.

Here we plot the results for all 50 runs from random initializations for each layer and architecture combinations to give a sense of the variance across runs. These runs are post-hyperoptimization and thus all use the same set of optimal hyperparameters. **Figure 10** shows the error on all pixels, **Figure 11** on the path pixels, and **Figure 12** on the distractor pixels.

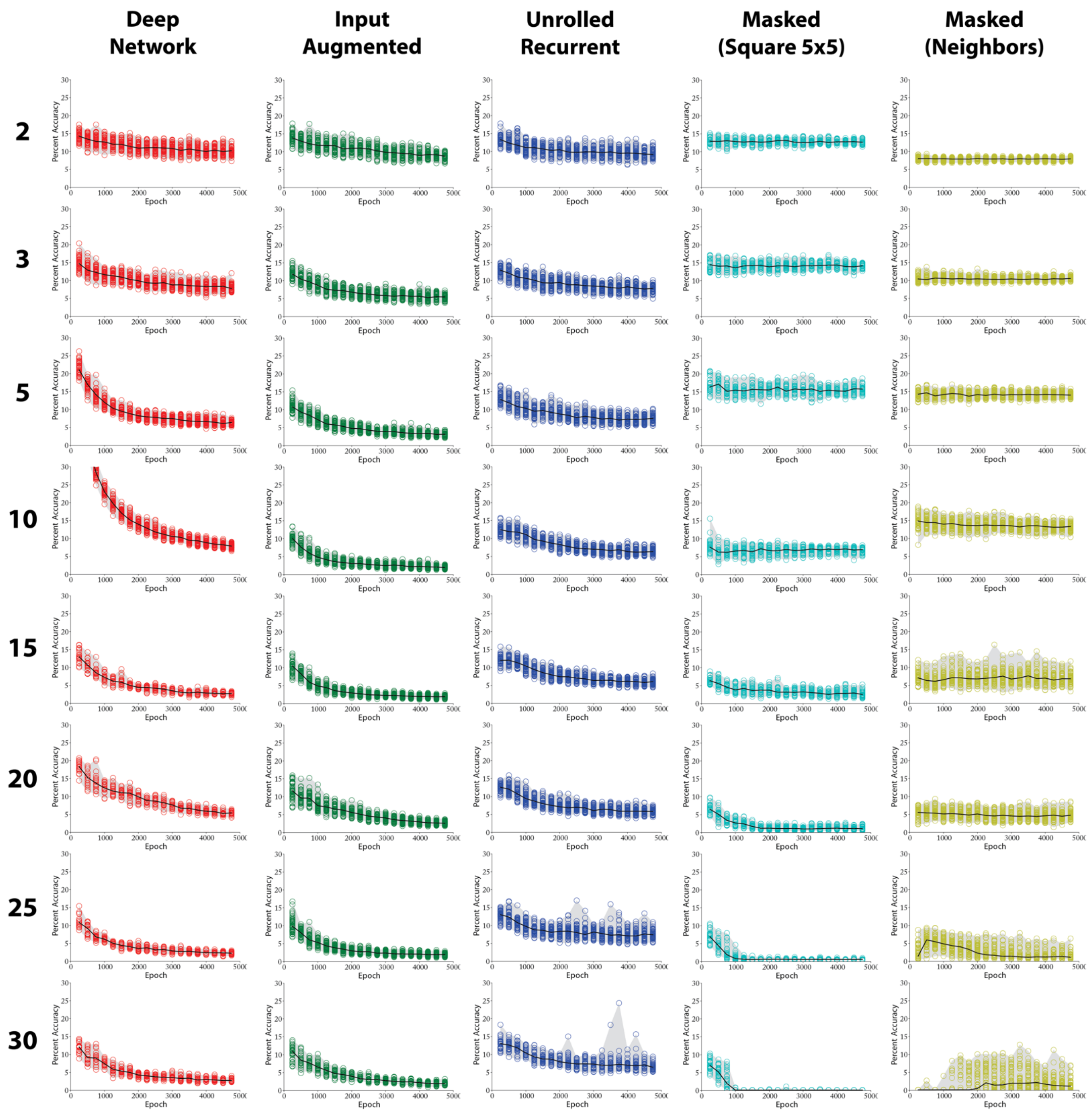


Fig S4. Path Pixel Error on All Runs. For each layer/architecture combination, 50 models were trained from random initializations for 50 epochs following hyperoptimization. Here we plot the error on path pixels every 250 epochs. The gray shading indicates the range of accuracies across all instantiations.

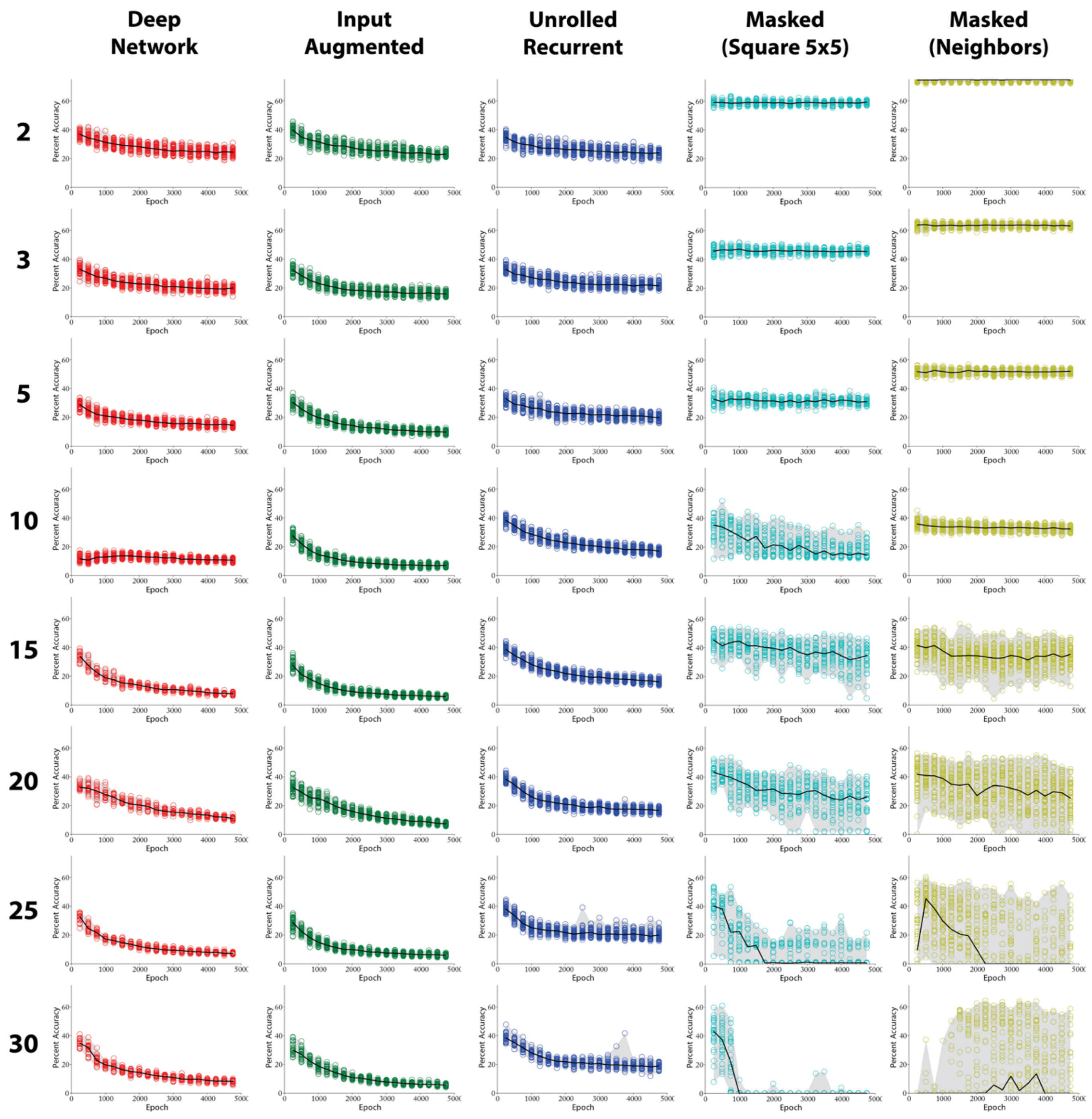


Fig S5. **Distractor Pixel Error on All Runs.** For each layer/architecture combination, 50 models were trained from random initializations for 50 epochs following hyperoptimization. Here we plot the error on distractor pixels every 250 epochs. The gray shading indicates the range of accuracies across all instantiations.

References

1. Cybenko G. Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems. 1989;2(4):303-14.
2. Girosi F, Poggio T. Networks and the best approximation property. Biological cybernetics. 1990;63(3):169-76.
3. Hornik K. Approximation capabilities of multilayer feedforward networks. Neural networks. 1991;4(2):251-7.
4. Lu Z, Pu H, Wang F, Hu Z, Wang L, editors. The expressive power of neural networks: A view from the width. Advances in neural information processing systems; 2017.
5. Rosenblatt F. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Cornell Aeronautical Lab Inc Buffalo NY; 1961.
6. Carandini M, Heeger DJ. Normalization as a canonical neural computation. Nature Reviews Neuroscience. 2012;13(1):51.
7. Srinivasan MV, Laughlin SB, Dubs A. Predictive coding: a fresh view of inhibition in the retina. Proceedings of the Royal Society of London Series B Biological Sciences. 1982;216(1205):427-59.
8. Rao RP, Ballard DH. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. Nature neuroscience. 1999;2(1):79-87.
9. Kapadia MK, Westheimer G, Gilbert CD. Spatial distribution of contextual interactions in primary visual cortex and in visual perception. Journal of neurophysiology. 2000;84(4):2048-62.
10. Stettler DD, Das A, Bennett J, Gilbert CD. Lateral connectivity and contextual interactions in macaque primary visual cortex. Neuron. 2002;36(4):739-50.
11. Boulanger-lewandowski N, Bengio Y, Vincent P, editors. Modeling temporal dependencies in highdimensional sequences: Application to polyphonic music generation and transcription. In ICML 29; 2012: Citeseer.
12. Graves A. Generating sequences with recurrent neural networks. arXiv preprint arXiv:13080850. 2013.
13. Chung J, Gulcehre C, Cho K, Bengio Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:14123555. 2014.
14. Sutskever I, Vinyals O, Le QV, editors. Sequence to sequence learning with neural networks. Advances in neural information processing systems; 2014.
15. Graves A, Mohamed A-r, Hinton G, editors. Speech recognition with deep recurrent neural networks. 2013 IEEE international conference on acoustics, speech and signal processing; 2013: IEEE.
16. Minsky M, Papert SA. Perceptrons: An introduction to computational geometry: MIT press; 2017.
17. Roelfsema PR, Singer W. Detecting connectedness. Cerebral cortex (New York, NY: 1991). 1998;8(5):385-96.
18. Roelfsema PR, BOHTE S, SPEKREIJSE H. Algorithms for the detection of connectedness and their neural implementation. Neuronal Information Processing: World Scientific; 1999. p. 81-103.
19. Lamme VA, Roelfsema PR. The distinct modes of vision offered by feedforward and recurrent processing. Trends in neurosciences. 2000;23(11):571-9.
20. Jehee JF, Lamme VA, Roelfsema PR. Boundary assignment in a recurrent network architecture. Vision research. 2007;47(9):1153-65.
21. Brosch T, Neumann H, Roelfsema PR. Reinforcement learning of linking and tracing contours in recurrent neural networks. PLoS computational biology. 2015;11(10).

22. LeCun Y, Bengio Y, Hinton G. Deep learning. nature. 2015;521(7553):436-44.
23. Goodfellow I, Bengio Y, Courville A. Deep learning: MIT press; 2016.
24. Hertz J, Krogh A, Palmer RG, Horner H. Introduction to the theory of neural computation. PhT. 1991;44(12):70.
25. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al., editors. Attention is all you need. Advances in neural information processing systems; 2017.
26. Veličković P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y, editors. Graph Attention Networks. International Conference on Learning Representations; 2018.
27. Yang Z, Yang D, Dyer C, He X, Smola A, Hovy E, editors. Hierarchical attention networks for document classification. Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies; 2016.
28. Nayebi A, Bear D, Kubilius J, Kar K, Ganguli S, Sussillo D, et al., editors. Task-driven convolutional recurrent models of the visual system. Advances in Neural Information Processing Systems; 2018.
29. Poggio T, Mhaskar H, Rosasco L, Miranda B, Liao Q. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. International Journal of Automation and Computing. 2017;14(5):503-19.
30. Linsley D, Kim J, Veerabadran V, Windolf C, Serre T, editors. Learning long-range spatial dependencies with horizontal gated recurrent units. Advances in Neural Information Processing Systems; 2018.
31. Kushnir L, Fusi S. Neural classifiers with limited connectivity and recurrent readouts. Journal of Neuroscience. 2018;38(46):9900-24.
32. Shallue CJ, Lee J, Antognini J, Sohl-Dickstein J, Frostig R, Dahl GE. Measuring the Effects of Data Parallelism on Neural Network Training. Journal of Machine Learning Research. 2019;20(112):1-49.
33. Maheswaranathan N, Williams A, Golub M, Ganguli S, Sussillo D, editors. Universality and individuality in neural dynamics across large populations of recurrent networks. Advances in neural information processing systems; 2019.
34. Metz L, Maheswaranathan N, Sun R, Freeman CD, Poole B, Sohl-Dickstein J. Using a thousand optimization tasks to learn hyperparameter search strategies. arXiv preprint arXiv:200211887. 2020.
35. Lee J, Schoenholz SS, Pennington J, Adlam B, Xiao L, Novak R, et al. Finite Versus Infinite Neural Networks: an Empirical Study. arXiv preprint arXiv:200715801. 2020.
36. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al., editors. PyTorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems; 2019.
37. Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980. 2014.
38. Li L, Jamieson K, DeSalvo G, Rostamizadeh A, Talwalkar A. Hyperband: A novel bandit-based approach to hyperparameter optimization. The Journal of Machine Learning Research. 2017;18(1):6765-816.
39. Bengio Y, Simard P, Frasconi P. Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks. 1994;5(2):157-66.
40. Pascanu R, Mikolov T, Bengio Y, editors. On the difficulty of training recurrent neural networks. International conference on machine learning; 2013.
41. He K, Zhang X, Ren S, Sun J, editors. Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition; 2016.