Inferring the function performed by a recurrent neural network

Matthew Chalk,[1] Gašper Tkačik,[2] and Olivier Marre[1]

[1]*Sorbonne Université, INSERM, CNRS, Institut de la Vision, 17 rue Moreau, F-75012 Paris, France*
[2]*IST Austria, A-3400, Klosterneuburg, Austria*

A central goal in systems neuroscience is to understand the functions performed by neural circuits. Previous top-down models addressed this question by comparing the behaviour of an ideal model circuit, optimised to perform a given function, with neural recordings. However, this requires guessing in advance what function is being performed, which may not be possible for many neural systems. Here, we propose an alternative approach that uses recorded neural responses to directly infer the function performed by a neural network. We assume that the goal of the network can be expressed via a reward function, which describes how desirable each state of the network is for carrying out a given objective. This allows us to frame the problem of optimising each neuron's responses by viewing neurons as agents in a reinforcement learning (RL) paradigm; likewise the problem of inferring the reward function from the observed dynamics can be treated using inverse RL. Our framework encompasses previous influential theories of neural coding, such as efficient coding and attractor network models, as special cases, given specific choices of reward function. Finally, we can use the reward function inferred from recorded neural responses to make testable predictions about how the network dynamics will adapt depending on contextual changes, such as cell death and/or varying input statistics, so as to carry out the same underlying function with different constraints.

Neural circuits have evolved to perform a range of different functions, from sensory coding to muscle control and decision making. A central goal of systems neuroscience is to elucidate what these functions are and how neural circuits implement them. A common 'top-down' approach starts by formulating a hypothesis about the function performed by a given neural system (e.g. efficient coding/decision making), which can be formalised via an objective function [1–10]. This hypothesis is then tested by comparing the predicted behaviour of a model circuit that maximises the assumed objective function (possibly given constraints, such as noise/metabolic costs etc.) with recorded responses.

One of the earliest applications of this approach was sensory coding, where neural circuits are thought to efficiently encode sensory stimuli, with limited information loss [7–13]. Over the years, top-down models have also been proposed for many central functions performed by neural circuits, such as generating the complex patterns of activity necessary for initiating motor commands [3], detecting predictive features in the environment [4], or memory storage [5]. Nevertheless, it has remained difficult to make quantitative contact between top-down model predictions and data, in particular, to rigorously test which (if any) of the proposed functions is actually being carried out by a real neural circuit.

The first problem is that a pure top-down approach requires us to hypothesise the function performed by a given neural circuit, which is often not possible. Second, even if our hypothesis is correct, there may be multiple ways for a neural circuit to perform the same function, so that the predictions of the top-down model may not match the data.

Here, we propose an approach to directly infer the function performed by a recurrent neural network from recorded responses. To do this, we first construct a top-down model describing how a recurrent neural network could optimally perform a range of possible functions, given constraints on how much information each neuron encodes about its inputs. A reward function quantifies how useful each state of the network is for performing a given function. The network's dynamics are then optimised to maximise how frequently the network visits states associated with a high reward. This framework is very general: different choices of reward function result in the network performing diverse functions, from efficient coding to decision making and optimal control.

Next, we tackle the inverse problem of infering the reward function from the observed network dynamics. To do this, we show that there is a direct correspondence between optimising a neural network as described above, and optimising an agent's actions (in this case, each neuron's responses) via reinforcement learning (RL) [14–18] (Fig 1). As a result, we can use *inverse* RL [19–23] to infer the objective performed by a network from recorded neural responses. Specifically, we are able to derive a closed-form expression for the reward function optimised by a network, given its observed dynamics.

We hypothesise that the inferred reward function, rather than e.g. the properties of individual neurons, is the most succinct mathematical summary of the network, that generalises across different contexts and conditions. Thus we can use our framework to quantitatively predict how the network will adapt or learn in order to perform the same function when the external context (e.g. stimulus statistics) constraints (e.g. noise level) or the structure of the network (e.g. due to cell death or experimental manipulation) change. Our framework thus not only allows RL to be used to train neural networks and use inverse RL to infer their function, but it also generates experimental predictions for a wide range of possible manipulations.

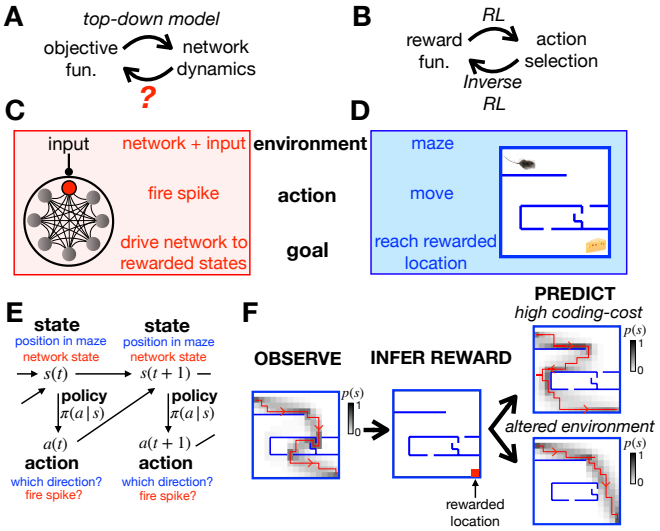

FIG. 1. **General approach.** (**A**) Top-down models use an assumed objective function to derive the optimal neural dynamics. Here we address the inverse problem, to infer the objective function from observed neural responses. (**B**) Similarly, RL uses an assumed reward function to derive an optimal set of actions that an agent should perform in a given environment, while inverse RL infers the reward function from the agent's actions. (**C-D**) A mapping between the neural network and textbook RL setup. (**E**) Both problems can be formulated as MDPs, where the agent (or neuron) can choose which actions, a, to perform to alter their state, s, and increase their reward. (**F**) Given a reward function and coding cost (which penalises complex policies), we can use entropy-regularised RL to derive the optimal policy (left). Here we plot a single trajectory sampled from the optimal policy (red), as well as how often the agent visits each location (shaded). Conversely, we can use inverse RL to infer the reward function from the agent's policy (centre). We can then use the inferred reward to predict how the agent's policy will change when we increase the coding cost to favour simpler (but less rewarded) trajectories (top right), or move the walls of the maze (bottom right).

RESULTS

General approach

We can quantify how well a network performs a specific function (e.g. sensory coding/decision making) via an objective function L_π (where π denotes the parameters that determine the network dynamics) (Fig 1A). There is a large literature describing how to optimise the dynamics of a neural network, π, so as to maximise a specific objective functions, L_π, given constraints (e.g. metabolic cost/wiring constraints etc.) [1–10]. However, it is generally much harder to go in the opposite direction, to infer the objective function, L_π, from observations of the neural network dynamics.

To address this question, we looked to the field of reinforcement learning (RL) [14–18], which describes how an agent should choose actions so as to maximise the reward they receive from their environment (Fig 1B). Conversely, another paradigm, called inverse RL [19–23], explains how to go in the opposite direction, to infer the reward associated with different states of the environment from observations of the agent's actions. We reasoned that, if we could establish a mapping between optimising neural network dynamics (Fig 1A) and optimising an agent's actions via RL (Fig 1B), then we could use inverse RL to infer the objective function optimised by a neural network from its observed dynamics.

To illustrate this idea, let us compare the problem faced by a single neuron embedded within a recurrent neural network (Fig 1C) to the textbook RL problem of an agent navigating a maze (Fig 1D). The neuron's environment is determined by the activity of other neurons in the network and its external input; the agent's environment is determined by the walls of the maze. At each time, the neuron can choose whether to fire a spike, so as to drive the network towards states that are 'desirable' for performing a given function; at each time, the agent in the maze can choose which direction to move in, so as to reach 'desirable' locations, associated with a high reward.

Both problems can be formulated mathematically as Markov Decision Processes (MDPs) (Fig 1E). Each state of the system, s (i.e. the agent's position in the maze, or the state of the network and external input), is associated with a reward, $r(s)$. At each time, the agent can choose to perform an action, a (i.e. moving in a particular direction, or firing a spike), so as to reach a new state s' with probability, $p(s'|a,s)$. The probability that the agent performs a given action in each state, $\pi(a|s)$, is called their policy.

We assume that the agent (or neuron) optimises their policy to maximise their average reward, $\langle r(s)\rangle_{p_\pi(s)}$, given a constraint on the information they can encode about their state, $I_\pi(a;s)$ (this corresponds, for example, to constraining how much a neuron can encode about the rest of the network and external input). This can be achieved by maximising the following objective function:

$$L_\pi = \langle r(s)\rangle_{p_\pi(s)} - \lambda I_\pi(a;s) \tag{1}$$

where λ is a constant that controls the strength of the constraint. Note that in the special case where the agent's state does not depend on previous actions (i.e. $p(s'|a,s) = p(s'|s)$) and the reward depends on their current state and action, this is the same as the objective function used in rate-distortion theory [24, 25]). We can also write the objective function as:

$$L_\pi = \langle r(s) - \lambda c_\pi(s)\rangle_{p_\pi(s)} \tag{2}$$

where $c_\pi(s)$ is a 'coding cost', equal to the Kullback-Leibler divergence between the agent's policy and the steady-state distribution over actions, $D_{KL}[\pi(a|s)\|p_\pi(a)]$. In SI section 1A we show how

this objective function can be maximised via entropy-regularised RL [15, 18] to obtain the optimal policy:

$$\pi^*(a|s) \propto p_\pi(a)\, e^{\frac{1}{\lambda}\left\langle v_{\pi^*}(s')\right\rangle_{p(s'|s,a)}}, \tag{3}$$

where $v_\pi(s)$ is a 'value function' defined as the predicted reward in the future (minus the coding cost) if the agent starts in each state (SI section 1A):

$$\begin{aligned} v_\pi(s) = r(s) - \lambda c_\pi(s) - L_\pi \\ + \left\langle r(s') - \lambda c_\pi(s')\right\rangle_{p_\pi(s'|s)} - L_\pi \\ + \left\langle r(s'') - \lambda c_\pi(s'')\right\rangle_{p_\pi(s''|s)} - L_\pi + \ldots \end{aligned} \tag{4}$$

where s, s' and s'' denote three consecutive states of the agent. Thus, actions that drive the agent towards high reward (and thus, high value) states are preferred over actions that drive the agent towards low reward (and thus, low value) states.

Let us return to our toy example of the agent in a maze. Figure 1F (left) shows the agent's trajectory through the maze after optimising their policy to maximise L_π. In this example, a single location, in the lower-right corner of the maze, has a non-zero reward (Fig 1F, centre). However, suppose we didn't know this; could we infer the reward at each location just by observing the agent's trajectory in the maze? In SI section 1A we show that this can be done by finding the reward function that maximises the log-likelihood of the optimal policy, averaged over observed actions and states, $\langle \log \pi^*(a|s)\rangle_{\text{data}}$. If the coding cost is non-zero ($\lambda > 0$), this problem is well-posed, meaning there is a unique solution for $r(s)$.

Once we know the reward function optimised by the agent, we can then use it to predict how their behaviour will change when we alter their external environment or internal constraints. For example, we can predict how the agent's trajectory through the maze will change when we move the position of the walls (Fig 1F, lower right), or increase the coding cost so as to favour simpler (but less rewarded) trajectories (Fig 1F, upper right).

Optimising neural network dynamics

We used these principles to infer the function performed by a recurrent neural network. We considered a model network of n neurons, each described by a binary variable, $\sigma_i = -1/1$, denoting whether the neuron is silent or spiking respectively (SI section 1B). The network receives an external input, $\boldsymbol{x}$. The network state is described by an n-dimensional vector of binary values, $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_n)^T$. Both the network and external input have Markov dynamics. Neurons are updated asynchronously: at each time-step a neuron is selected at random, and its state updated by sampling from $\pi_i(\tilde{\sigma}_i|\boldsymbol{\sigma}, \boldsymbol{x})$. The dynamics of the network are fully specified by the set of transition probabilities, $\pi_i(\tilde{\sigma}_i|\boldsymbol{\sigma}, \boldsymbol{x})$, and input statistics, $p(\boldsymbol{x}'|\boldsymbol{x})$.

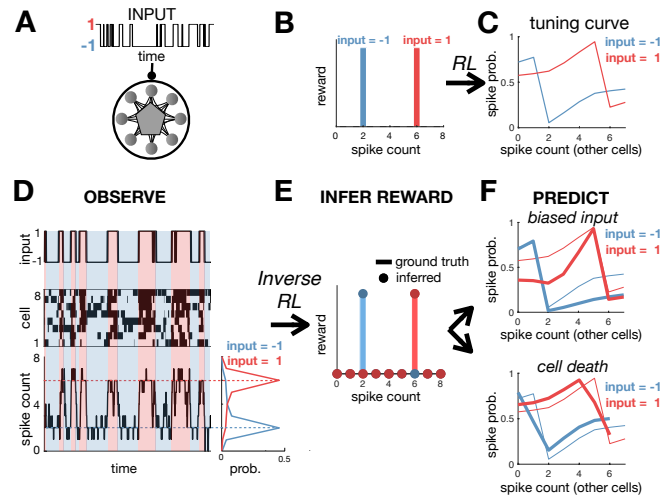

FIG. 2. **Inferring the function performed by a neural network.** (**A**) A recurrent neural network receives a binary input, x. (**B**) The reward function equals 1 if the network fires 2 spikes when $x = -1$, or 6 spikes when $x = 1$. (**C**) The optimal neural tuning curves depend on the input, x, and total spike count. (**D**) Simulated dynamics of the network with 8 neurons (left). The total spike count (below) is tightly peaked around the rewarded values. (**E**) Using inverse RL, we infer the original reward function used to optimise the network from its observed dynamics. (**F**) The inferred reward function is used to predict how neural tuning curves will adapt depending on contextual changes, such as varying the input statistics (e.g. decreasing $p(x = 1)$) (top right), or cell death (bottom right). Thick/thin lines show adapted/original tuning curves, respectively.

As before, we use a reward function, $r(\boldsymbol{\sigma}, \boldsymbol{x})$, to express how desirable each state of the network is to perform a given functional objective. For example, if the objective of the network is to faithfully encode the external input, then an appropriate reward function might be the negative squared error: $r(\boldsymbol{x}, \boldsymbol{\sigma}) = -(\boldsymbol{x} - \hat{\boldsymbol{x}}(\boldsymbol{\sigma}))^2$, where $\hat{\boldsymbol{x}}(\boldsymbol{\sigma})$ denotes an estimate of $\boldsymbol{x}$, inferred from the network state, $\boldsymbol{\sigma}$. More generally, different choices of reward function can be used to describe a large range of functions that may be performed by the network.

The dynamics of the network, π, are said to be optimal if they maximise the average reward, $\langle r(\boldsymbol{\sigma}, \boldsymbol{x})\rangle_{p_\pi(\boldsymbol{\sigma}, \boldsymbol{x})}$, given a constraint on the information each neuron encodes about the rest of the network and external inputs, $\sum_{i=1}^n I_\pi(\tilde{\sigma}_i; \boldsymbol{\sigma}, \boldsymbol{x})$. This corresponds to maximising the objective function:

$$L_\pi = \left\langle r(\boldsymbol{\sigma}, \boldsymbol{x})\right\rangle_{p_\pi(\boldsymbol{\sigma}, \boldsymbol{x})} - \lambda \sum_{i=1}^n I_\pi(\tilde{\sigma}_i; \boldsymbol{\sigma}, \boldsymbol{x}). \tag{5}$$

where λ controls the strength of the constraint.

For each neuron, we can frame this optimisation problem as an MDP, where the state, action, and policy correspond to the network state and external input $\{\boldsymbol{\sigma}, \boldsymbol{x}\}$, the neuron's proposed update $\tilde{\sigma}_i$, and the transition probability $\pi_i(\tilde{\sigma}_i|\boldsymbol{\sigma}, \boldsymbol{x})$, respectively. Thus, we can use entropy-

regularised RL to optimise each neuron's response probability, $\pi_i(\tilde{\sigma}_i|\boldsymbol{\sigma},\boldsymbol{x})$, exactly as we did for the agent in the maze. Further, as each update increases the objective function L_π, we can alternate updates for different neurons to optimise the dynamics of the entire network. In SI section 1B, we show that this results in transition probabilities given by:

$$\pi_i^*\left(\tilde{\sigma}_i = 1|\boldsymbol{\sigma}_{/i},\boldsymbol{x}\right) = \left(1 + \frac{p_{\pi^*}(\sigma_i)}{1-p_{\pi^*}(\sigma_i)} e^{\frac{1}{n\lambda}\delta v_{\pi^*}(\boldsymbol{\sigma}_{/i},\boldsymbol{x})}\right)^{-1}, \tag{6}$$

where $\boldsymbol{\sigma}_{/i}$ is the state of all neurons except neuron i, and

$$\delta v_{\pi^*}\left(\boldsymbol{\sigma}_{/i},\boldsymbol{x}\right) = \sum_{\sigma_i=-1,1} \sigma_i \left\langle v_{\pi^*}\left(\boldsymbol{\sigma},\boldsymbol{x}'\right)\right\rangle_{p(\boldsymbol{x}'|\boldsymbol{x})}, \tag{7}$$

where $v_\pi(\boldsymbol{\sigma},\boldsymbol{x})$ is the value associated with each state, defined in Eqn 4, with a coding cost, $c_\pi(\boldsymbol{\sigma},\boldsymbol{x}) = \sum_{i=1}^n D_{KL}\left[\pi_i(\tilde{\sigma}_i|\boldsymbol{\sigma},\boldsymbol{x})\|p_\pi(\tilde{\sigma}_i)\right]$, which penalises deviations from each neuron's average firing rate. The network dynamics are optimised by alternately updating the value function and neural response probabilities (SI section 1B).

To see how this works in practice, we simulated a network of 8 neurons that receive a binary input x (Fig 2A). The assumed goal of the network is to fire exactly 2 spikes when $x = -1$, and 6 spikes when $x = 1$, while minimising the coding cost. To achieve this, the reward was set to unity when the network fired the desired number of spikes, and zero otherwise (Fig 2B). Using entropy-regularised RL, we derived optimal tuning curves for each neuron, which show how their spiking probability should vary depending on the input, x, and number of spikes fired by other neurons (Fig 2C). We confirmed that after optimisation the number of spikes fired by the network was tightly peaked around the target values (Fig 2D). Decreasing the coding cost reduced noise in the network, decreasing variability in the total spike count.

Inferring the objective function from the neural dynamics

We next asked if we could use inverse RL to infer the reward function optimised by a neural network, just from its observed dynamics (Fig 2D). For simplicity, we first consider a recurrent network that receives no external input, in which case the optimal dynamics (Eqn 6) correspond to Gibbs sampling from a steady-state distribution: $p(\boldsymbol{\sigma}) \propto \prod_i p(\sigma_i)\exp\left(\frac{1}{\lambda n} v_\pi(\boldsymbol{\sigma})\right)$. Combining this with the Bellmann equality (which relates the value function, $v_\pi(\boldsymbol{\sigma},\boldsymbol{x})$, to the reward function, $r(\boldsymbol{\sigma},\boldsymbol{x})$), we can derive the following expression for the reward function:

$$r(\boldsymbol{\sigma}) \propto \sum_{i=1}^{i=n} \log\left(\frac{p\left(\sigma_i|\boldsymbol{\sigma}_{/i}\right)}{p(\sigma_i)}\right) + C, \tag{8}$$

where $p\left(\sigma_i|\boldsymbol{\sigma}_{/i}\right)$ denotes the probability that neuron i is in state σ_i, given the current state of all the other neurons, C is an irrelevant constant, and the unknown constant of proportionality comes from the fact that we don't know the coding cost, λ. SI section 1C shows how we can recover the reward function when there is an external input. In this case, we do not obtain a closed-form expression for the reward function, but must instead infer it via maximum likelihood.

Figure 2E shows how we can use inverse RL to infer the reward function optimised by a model network from its observed dynamics. As for the agent in the maze, once we know the reward function, we can use it to predict how the network dynamics will vary depending on the internal/external constraints. For example, we can predict how neural tuning curves vary when we alter the input statistics (Fig 2F, upper), or kill a cell (Fig 2F, lower).

Application to a pair-wise coupled network

Our framework is limited by the fact that the number of states, n_s, scales exponentially with the number of neurons ($n_s = 2^n$). Thus, it quickly becomes infeasible to compute the optimal dynamics as the number of neurons increases. Likewise, we need an exponential amount of data to reliably estimate the sufficient statistics of the network, required to infer the reward function.

This problem can be circumvented by using a tractable parametric approximation of the value function. For example, if we approximate the value function by a quadratic function of the responses our framework predicts a steady-state response distribution of the form: $p(\boldsymbol{\sigma}) \propto \exp\left(\sum_{i,j\neq i} J_{ij}\sigma_i\sigma_j + \sum_i h_i\sigma_i\right)$, where J_{ij} denotes the pair-wise coupling between neurons, and h_i is the bias. This corresponds to a pair-wise Ising model, which has been used previously to model recorded neural responses [26, 27]. In SI section 1D we derive an approximate RL algorithm to optimise the coupling matrix, $\boldsymbol{J}$, for a given reward function $r(\boldsymbol{\sigma})$ and coding cost.

To illustrate this, we simulated a network of 12 neurons arranged in a ring, with reward function equal to 1 if exactly 4 adjacent neurons are active together, and 0 otherwise. After optimisation, nearby neurons had positive couplings, while distant neurons had negative couplings (Fig 3A). The network dynamics generated a single hill of activity which drifted smoothly in time. This is reminiscent of ring attractor models, which have been influential in modeling neural functions such as the rodent head direction system [28, 29].

As before, we could then use inverse RL to infer the reward function from the observed network dynamics. However, note that when we use a parametric approximation of the value function this problem is not well-posed, and we have to make additional assumptions (SI section 1D). In our simulation, we found that assuming that the reward function was both sparse and positive was sufficient to recover the original reward function used to

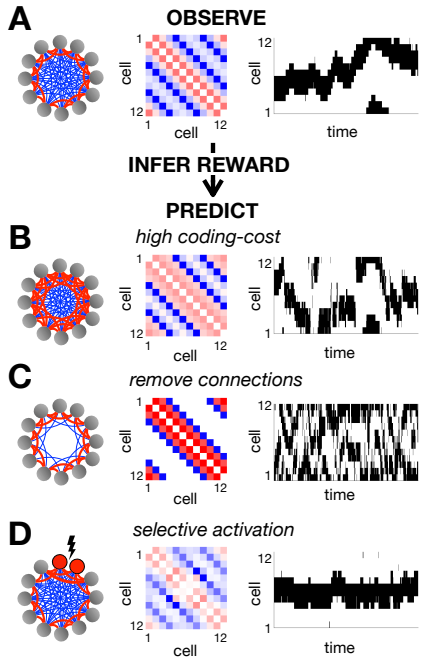

FIG. 3. **Pair-wise coupled network.** (**A**) We optimized the parameters of a pair-wise coupled network, using a reward function that was equal to 1 when exactly 4 adjacent neurons were simultaneously active, and 0 otherwise. The resulting couplings between neurons are schematized on the left, with positive couplings in red and negative couplings in blue. The exact coupling strengths are plotted in the centre. On the right we show an example of the network dynamics. Using inverse RL, we can infer the original reward function used to optimise the network from its observed dynamics. We can then use this inferred reward to predict how the network dynamics will vary when we increase the coding cost (**B**), remove connections between distant neurons (**C**) or selectively activate certain neurons (**D**).

optimise the network (SI section 2C).

Having inferred the reward function optimised by the network, we can use it to predict how the coupling matrix, $\boldsymbol{J}$, and network dynamics vary depending on the internal/external constraints. For example, increasing the coding cost resulted in stronger positive couplings between nearby neurons and a hill of activity that sometimes jumped discontinuously between locations (Fig 3B); removing connections between distant neurons resulted in two uncoordinated peaks of activity (Fig 3C); finally, selectively activating certain neurons 'pinned' the hill of activity to a single location (Fig 3D).

Inferring efficiently encoded stimulus features

An influential hypothesis, called 'efficient coding', posits that sensory neural circuits have evolved to encode maximal information about sensory stimuli, given internal constraints [7–13]. However, the theory does not specify which stimulus features are relevant to the organism, and thus should be encoded. Here we show how one can use inverse RL to: (i) infer which stimulus features are encoded by a recorded neural network, and (ii) test whether these features are encoded efficiently.

Efficient coding posits that neurons maximise information encoded about some relevant feature, $\boldsymbol{y}(\boldsymbol{x})$, given constraints on the information encoded by each neuron about their inputs, $\boldsymbol{x}$ (Fig 4A). This corresponds to maximising:

$$L_\pi = I_\pi\left(\boldsymbol{y}(\boldsymbol{x});\boldsymbol{\sigma}\right) - \lambda \sum_{i=1}^{n} I_\pi\left(\tilde{\sigma}_i;\boldsymbol{\sigma},\boldsymbol{x}\right), \tag{9}$$

where λ controls the strength of the constraint. Noting that the second term is equal to the coding cost we used previously (Eqn 5), we can rewrite this objective function as:

$$L_\pi = \left\langle \log p_\pi\left(\boldsymbol{y}(\boldsymbol{x})\,|\boldsymbol{\sigma}\right) - \lambda c_\pi\left(\boldsymbol{\sigma},\boldsymbol{x}\right)\right\rangle_{p_\pi(\boldsymbol{\sigma}|\boldsymbol{x})p(\boldsymbol{x})} \tag{10}$$

where we have omitted terms which don't depend on π. Now this is exactly the same as the objective function we have been using so far (Eqn 5), in the special case where the reward function, $r(\boldsymbol{\sigma},\boldsymbol{x})$, is equal to the log-posterior, $\log p_\pi\left(\boldsymbol{y}(\boldsymbol{x})|\boldsymbol{\sigma}\right)$. As a result we can maximise L_π via an iterative algorithm, where on each iteration we update the reward function by setting $r(\boldsymbol{x},\boldsymbol{\sigma}) \leftarrow \log p_\pi(\boldsymbol{y}(\boldsymbol{x})\,|\boldsymbol{\sigma})$, before then optimising the network dynamics, via entropy-regularised RL. Thus, thanks to the correspondence between entropy-regularised RL and efficient coding we could derive an algorithm to optimise the dynamics of a recurrent network to perform efficient coding [27].

As an illustration, we simulated a network of 7 neurons that receive a sensory input consisting of 7 binary pixels (Fig 4B, top). Here, the 'relevant feature', $y(\boldsymbol{x})$ was a single binary variable, which was equal to 1 if more than 4 pixels were active, and -1 otherwise (Fig 4A, bottom). Using the efficient-coding algorithm described above, we derived optimal tuning curves, showing how each neuron's spiking probability should vary with both the number of active pixels and number of spikes fired by other neurons (Fig 4C). We also derived how the optimal readout, $p(y|\boldsymbol{\sigma})$, depended on the number of spiking neurons (Fig 4D). Finally, we checked that the optimised network encoded significantly more information about the relevant feature than a network of independent neurons, over a large range of coding costs (Fig 4E).

Now, imagine that we just observe the stimulus and neural responses (Fig 4F). Can we recover the relevant feature, $y(\boldsymbol{x})$? To do this, we first use inverse RL to infer the reward function from observed neural responses (Fig 4G). As described above, if the network is performing efficient coding then the inferred reward, $r(\boldsymbol{\sigma},\boldsymbol{x})$ should be proportional to the log-posterior, $\log p\left(y(\boldsymbol{x})|\boldsymbol{\sigma}\right)$. Thus, given $\boldsymbol{\sigma}$, the inferred reward, $r(\boldsymbol{\sigma},\boldsymbol{x})$ should only depend on changes to the input, $\boldsymbol{x}$, that alter $y(\boldsymbol{x})$. As a result, we can use the inferred reward to uncover all inputs, $\boldsymbol{x}$, that map onto the same

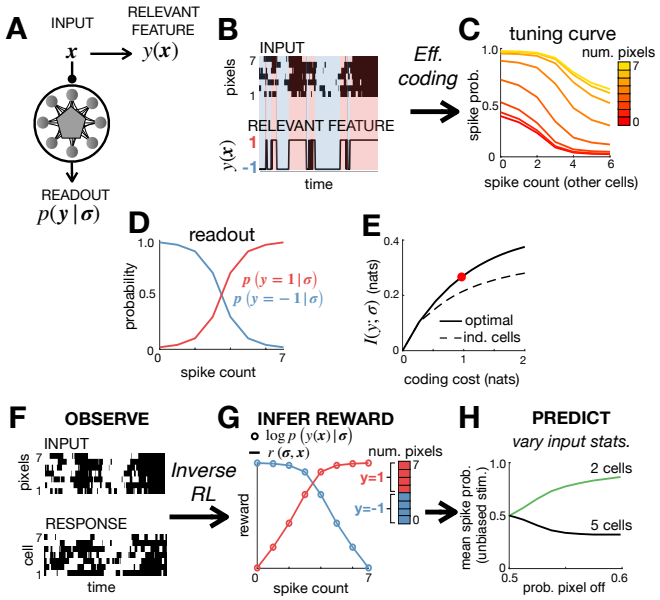

FIG. 4. **Efficient coding and inverse RL.** (**A**) The neural code was optimised to efficiently encode an external input, $\boldsymbol{x}$, so as to maximise information about a relevant stimulus feature $y\,(\boldsymbol{x})$. (**B**) The input, $\boldsymbol{x}$ consisted of 7 binary pixels. The relevant feature, $y\,(\boldsymbol{x})$, was equal to 1 if >4 pixels were active, and -1 otherwise. (**C**) Optimising a network of 7 neurons to efficiently encode $y\,(\boldsymbol{x})$ resulted in all neurons having identical tuning curves, which depended on the number of active pixels and total spike count. (**D**) The posterior probability that $y = 1$ varied monotonically with the spike count. (**E**) The optimised network encoded significantly more information about $y\,(\boldsymbol{x})$ than a network of independent neurons with matching stimulus-dependent spiking probabilities, $p\,(\sigma_i = 1|\boldsymbol{x})$. The coding cost used for the simulations in the other panels is indicated by a red circle. (**F-G**) We use the observed responses of the network (**F**) to infer the reward function optimised by the network, $r\,(\boldsymbol{\sigma}, \boldsymbol{x})$ (**G**). If the network efficiently encodes a relevant feature, $y\,(\boldsymbol{x})$, then the inferred reward (solid lines) should be proportional to the log-posterior, $\log p\,(y\,(\boldsymbol{x})|\boldsymbol{\sigma})$ (empty circles). This allows us to (i) recover $y\,(\boldsymbol{x})$ from observed neural responses, (ii) test whether this feature is encoded efficiently by the network. (**H**) We can use the inferred objective to predict how varying the input statistics, by reducing the probability that pixels are active, causes the population to split into two cell types, with different tuning curves and mean firing rates (right).

value of $y\,(\boldsymbol{x})$. Here, we see that the inferred reward collapses onto two curves only (blue and red in Fig 4G), depending on the total number of pixels in the stimulus. This allows us to deduce that the relevant coded variable, $y\,(\boldsymbol{x})$, must be a sharp threshold on the number of simultaneously active pixels (despite the neural tuning curves varying smoothly with the number of active pixels; Fig 4C). Next, having recovered $y\,(\boldsymbol{x})$, we can check whether it is encoded efficiently by seeing whether the inferred reward, $r\,(\boldsymbol{\sigma}, \boldsymbol{x})$ is proportional to the log-posterior, $\log p\,(y\,(\boldsymbol{x})|\boldsymbol{\sigma})$.

Finally, once we have inferred the function performed by the network, we can predict how its dynamics will vary with context, such as when we alter the input statistics. For example, in our simulation, reducing the probability that input pixels are active causes the neural population to split into two cell-types, with distinct tuning curves and mean firing rates (Fig 4H) [13].

DISCUSSION

A large research effort has been devoted to developing 'top-down' models, which describe the network dynamics required to optimally perform a given function (e.g. decision making [6], control [3], efficient sensory coding [8] etc.). Here, we go in the opposite direction: starting from recorded responses, we show how to infer the objective function optimised by the network. This approach, where the objective function is inferred from data, should allow one to: (i) quantitatively predict the responses of a given recorded network, from functional principles; (ii) build top-down models of high-level neural areas, whose function is *a priori* unknown.

An alternative bottom-up approach is to construct phenomenological models describing how neurons respond to given sensory stimuli and/or neural inputs [26, 27, 30, 31]. In common with our work, such models are directly fitted to neural data. However, unlike our work, they do not set out to reveal the function performed by the network. Further, they are often poor at predicting neural responses in different contexts (e.g. varying stimulus statistics). Here we hypothesize that it is the function performed by a neural circuit that remains invariant, not its dynamics or individual cell properties. Thus, if we can infer what this function is, we should be able to predict how the network dynamics will adapt depending on the context, so as to perform the same function under different constraints. As a result, our theory could predict how the dynamics of a recorded network will adapt in response to a large range of experimental manipulations, such as varying the stimulus statistics, blocking connections, knocking out/stimulating cells etc.

There is an extensive literature on how neural networks could perform RL [32–34]. Our focus here was different: we sought to use tools from RL and inverse RL to infer the function performed by a recurrent neural network. Thus, we do not assume the network receives an explicit reward signal: the reward function is simply a way of expressing which states of the network are useful for performing a given function. In contrast to previous work, we treat each neuron as an independent agent, which optimises their responses to maximise the reward achieved by the network, given a constraint on how much they can encode about their inputs. As well as being required for biological realism, the coding constraint has the benefit of making the inverse RL problem well-posed. Indeed, under certain assumptions, we show that it is possible to write a closed form expression for the reward function optimised by the network, given its steady-state distri-

bution (Eqn 8).

Our framework relies on several assumptions about the network dynamics. First, we assume that the network has Markov dynamics, such that its state depends only on the preceding time-step. To relax this assumption, we could redefine the network state to include spiking activity in several time-steps. For example, we could thus include the fact that neurons are unlikely to fire two spikes within a given temporal window, called their refractory period. Of course, this increase in complexity would come at the expense of decreased computational tractability, which may necessitate approximations. Second, we assume the only constraint that neurons face is a 'coding cost', which limits how much information they encode about other neurons and external inputs. In reality, biological networks face many other constraints, such as the metabolic cost of spiking and constraints on the wiring between cells, which we may want to incorporate explicitly into our framework in the future.

Our work unifies several influential theories of neural coding, that were considered separately in previous work. For example, we show a direct link between entropy-regularised RL [15–18] (Fig 1), ring-attractor networks [28, 29] (Fig 3), and efficient sensory coding [7–13] (Fig 4). Further, given a static network without dynamics, our framework is directly equivalent to rate-distortion theory [24, 25]. While interesting in its own right, this generality means that we can potentially apply our theory to infer the function performed by diverse neural circuits, that have evolved to perform a broad range of different functional objectives. This contrasts with previous work, where neural data is typically used to test a single top-down hypothesis, formulated in advance.

ACKNOWLEDGMENTS

This work was supported by ANR JCJC grant (ANR-17-CE37-0013) to M.C, ANR Trajectory (ANR-15-CE37-0011), ANR DECORE (ANR-18-CE37-0011), the French State program Investissements d'Avenir managed by the Agence Nationale de la Recherche (LIFESENSES; ANR-10-LABX-65), EC Grant No. H2020-785907 from the Human Brain Project (SGA2), and an AVIESAN-UNADEV grant to O.M. The authors would like to thank Ulisse Ferrari for useful discussions and feedback.

[1] Yang G R, Joglekar MR, Song HF, Newsome WT, Wang XJ. (2019). Task representations in neural networks trained to perform many cognitive tasks. *Nat Neurosci*, 22:297–306

[2] Heeger D J (2017) Theory of cortical function. *Proc Natl Acad Sci USA* 114:1773-1782

[3] Sussillo D, Abbott L F (2009) Generating coherent patterns of activity from chaotic neural networks. *Neuron* 63:544-557.

[4] Gutig R (2016) Spiking neurons can discover predictive features by aggregate–label learning. *Science* 351(6277):aab4113

[5] Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational abilities *Proc Natl Acad Sci USA* 79:2554–2558

[6] Krding K (2007) Decision theory: what should the nervous system do? *Science* 318:606-610

[7] Boerlin M, Machens CK, Denve S (2013) Predictive coding of dynamical variables in balanced spiking networks. *PLoS Comp Bio* 9 e1003258.

[8] Simoncelli EP, Olshausen BA (2001) Natural image statistics and neural representation. *Ann Rev Neurosci* 24:1193-1216

[9] Tkacik G, Prentice JS, Balasubramanian V, Schneidman E (2010) Optimal population coding by noisy spiking neurons. *Proc Natl Acad Sci USA* 107:14419-14424.

[10] Chalk M, Marre O, Tkaik G (2018) Toward a unified theory of efficient, predictive, and sparse coding. *Proc Natl Acad Sci USA* 115:186–191.

[11] Barlow, HB (1961) Possible principles underlying the transformations of sensory messages. *Sensory Communication,* ed Rosenblith WA (MIT Press, Cambridge, MA), pp 217–234

[12] Field DJ (1994) What is the goal of sensory coding? *Neural Comput* 6:559–601.

[13] Gjorgjieva J, Sompolinsky H, Meister M (2014) Benefits of pathway splitting in sensory coding. *J Neurosci* 34:12127-12144.

[14] Sutton RS, Barto AG (2018) Reinforcement learning: An introduction. MIT press.

[15] Todorov E (2008) General duality between optimal control and estimation. *Proc of the 47th IEEE Conference on Decision and Control* 4286-4292

[16] Schulman J, Chen X, Abbeel P (2017) Equivalence between policy gradients and soft Q-learning. arXiv:1704.06440.

[17] Haarnoja T, Tang H, Abbeel P, Levine S (2017). Reinforcement learning with deep energy-based policies. *Proc 34th International Conf on Machine Learning* 70:1352-1361

[18] Tiomkin S, Tishby N (2017). A Unified Bellman Equation for Causal Information and Value in Markov Decision Processes. arXiv:1703.01585.

[19] Ng AY, Russell SJ (2000) Algorithms for inverse reinforcement learning. *Proc of the 17th International Con on Machine Learning* 663–670

[20] Rothkopf CA, Dimitrakakis C (2011) Preference elicitation and inverse reinforcement learning. In *Joint European conference on machine learning and knowledge discovery in databases* Springer pp. 34–48.

[21] Herman M, Gindele T, Wagner J, Schmitt F, Burgard W (2016) Inverse reinforcement learning with simultaneous estimation of rewards and dynamics. *Artificial Intelligence and Statistics* 102–110

[22] Wu Z, Schrater P, Pitkow X (2018) Inverse POMDP: Inferring What You Think from What You Do. arXiv:1805.09864.

[23] Reddy S, Dragan AD, Levine S (2018) Where Do You

Think You're Going?: Inferring Beliefs about Dynamics from Behavior. arXiv:1805.08010.

[24] Berger T. Rate Distortion Theory. (1971) Englewood Clis.

[25] Bialek W, van Steveninck RRDR, Tishby N (2006) Efficient representation as a design principle for neural coding and computation. *IEEE international symposium on information theory* 659–663

[26] Schneidman E, Berry MJ, Segev R, Bialek W (2006) Weak pairwise correlations imply strongly correlated network states in a neural population. *Nature* 440:1007–1012

[27] Tkacik G, Marre O, Amodei D, Schneidman E, Bialek W, Berry MJ (2014) Searching for collective behavior in a large network of sensory neurons. *PLoS Comp Bio* 10:e1003408.

[28] Ben-Yishai R, Bar-Or RL, Sompolinsky H (1995) Theory of orientation tuning in visual cortex. *Proc Natl Acad Sci*, 92:3844–3848

[29] Zhang K (1996) Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: a theory. *J Neurosci* 16:2112–2126.

[30] Pillow JW, Shlens J, Paninski L, Sher A, Litke AM, Chichilnisky EJ, Simoncelli EP (2008) Spatio-temporal correlations and visual signalling in a complete neuronal population. *Nature* 454:995–999

[31] McIntosh L, Maheswaranathan N, Nayebi A, Ganguli S, Baccus S (2016) Deep learning models of the retinal response to natural scenes. *Adv Neur Inf Proc Sys* 29:1369–1377

[32] Niv Y (2009) Reinforcement learning in the brain. *J Mathemat Psychol* 53:139–154

[33] Dayan P, Niv Y (2008) Reinforcement learning: the good, the bad and the ugly. *Curr Op Neurobio* 18:185-196.

[34] Daw ND, Doya K (2006) The computational neurobiology of learning and reward. *Curr Op Neurobio* 16:199-204.

Inferring the function performed by a recurrent neural network
Supplementary information

Matthew Chalk,[1] Gašper Tkačik,[2] and Olivier Marre[1]

[1] *Sorbonne Université, INSERM, CNRS, Institut de la Vision, 17 rue Moreau, F-75012 Paris, France*
[2] *IST Austria, A-3400, Klosterneuburg, Austria*

I. SUPPLEMENTARY THEORY

A. Entropy-regularised RL

We consider a Markov Decision Process (MDP). Each state of the agent s, is associated with a reward, $r(s)$. At each time, the agent performs an action, a, sampled from a probability distribution, $\pi(a|s)$, called their policy. A new state, s', then occurs with a probability, $p(s'|s,a)$.

We seek a policy, $\pi\left(a|s\right)$, that maximises the average reward, constrained on the mutual information between between actions and states. This corresponds to maximising the Lagrangian:

$$L_\pi = \left\langle r\left(s\right)\right\rangle_{p_\pi(s)} - \lambda I_\pi\left(A;S\right) \tag{1}$$

$$= \left\langle r\left(s\right) - \lambda c_\pi\left(s\right)\right\rangle_{p_\pi(s)} \tag{2}$$

where λ is a lagrange-multiplier that determines the strength of the constraint, and $c_\pi\left(s\right) = D_{KL}\left[\pi\left(a|s\right)\|p(a)\right]$.

Now, let us can define a value function:

$$\begin{aligned} v_\pi\left(s\right) = r\left(s\right) - \lambda c_\pi\left(s\right) - L_\pi \\ + \left\langle r\left(s'\right) - \lambda c_\pi\left(s'\right)\right\rangle_{p_\pi(s'|s)} - L_\pi \\ + \left\langle r\left(s''\right) - \lambda c_\pi\left(s''\right)\right\rangle_{p_\pi(s''|s)} - L_\pi + \ldots \end{aligned} \tag{3}$$

where s, s' and s'' denote the agent's state in three consecutive time-steps. We can write the following Bellmann equality for the value function:

$$v_\pi(s) = r(s) - \lambda c_\pi(s) - L_\pi + \left\langle v_\pi\left(s'\right)\right\rangle_{p_\pi(s'|s)}. \tag{4}$$

Now, given an initial policy π^{old}, one can show, via the policy improvement theorem [1–4], that applying the greedy policy update,

$$\pi^{\text{new}}\left(a|s\right) = \arg\max_\pi \left(-\lambda c_\pi(s) + \left\langle v_{\pi^{\text{old}}}\left(s'\right)\right\rangle_{p_\pi(s'|s)}\right), \tag{5}$$

is guaranteed to improve the objective function, so that $L_{\pi^{\text{new}}} \geq L_{\pi^{\text{old}}}$. Solving the above maximisation, we arrive at the following update:

$$\pi^{\text{new}}(a|s) = \frac{1}{Z_{\pi^{\text{old}}}\left(s\right)} p_{\pi^{\text{old}}}(a) e^{\frac{1}{\lambda}\left\langle v_{\pi^{\text{old}}}\left(s'\right)\right\rangle_{p(s'|s,a)}} \tag{6}$$

where $Z_{\pi^{\text{old}}}\left(s\right)$ is a normalisation constant. Repeated application of the Bellmann recursion (Eqn 4) and greedy policy update (Eqn 6) returns the optimal policy, $\pi^*(a|s)$, which maximises L_π.

1. *Inverse entropy-regularized RL*

We can write the Bellmann recursion in Eqn 4 in vector form:

$$\boldsymbol{v} = \boldsymbol{r} - \lambda \boldsymbol{c} - L_\pi + \boldsymbol{P}\boldsymbol{v} \tag{7}$$

where $\boldsymbol{v}$, $\boldsymbol{c}$ and $\boldsymbol{r}$ are vectors with elements, $r_s \equiv r\left(s\right)$, $v_s \equiv v\left(s\right)$, $\boldsymbol{P}$ is a matrix with elements $P_{ss'} = \sum_a p\left(s'|a,s\right)\pi\left(a|s\right)$. Solving for $\boldsymbol{v}$, we have:

$$\boldsymbol{v} = \left(\boldsymbol{I} - \boldsymbol{P}\right)^{-1}\left(\boldsymbol{r} - \lambda \boldsymbol{c}\right) \tag{8}$$

Substituting this expression into Eqn 6, gives an expression for the optimal policy as a function of the reward:

$$\boldsymbol{\pi}_a^* \propto p(a)\exp\left(\boldsymbol{P}_a\left(\boldsymbol{I} - \boldsymbol{P}\right)^{-1}\left(\boldsymbol{r} - \lambda \boldsymbol{c}\right)\right) \tag{9}$$

where $\boldsymbol{\pi}_a$ is a vector with elements, $\left(\boldsymbol{\pi}_a\right)_s \equiv \pi\left(a|s\right)$ and $\boldsymbol{P}_a$ is a matrix, with elements $\left(\boldsymbol{P}_a\right)_{ss'} \equiv p\left(s'|a,s\right)$.

To infer the reward function, $r\left(s\right)$, we use the observed policy, $\pi\left(a|s\right)$ and transition probabilities, $p\left(s'|a,s\right)$, to estimate $\boldsymbol{c}$ and $\boldsymbol{P}$, $\boldsymbol{P}_a$ and $p\left(a\right)$. We then perform numerical optimisation to find the reward that maximises the log-likelihood of the optimal policy in Eqn 9, $\left\langle \log \pi^*\left(a|s\right)\right\rangle_{\mathcal{D}}$, averaged over data, $\mathcal{D}$.

B. Optimising a neural network via RL

We consider a recurrent neural network, with n neurons, each described by a binary variable, $\sigma_i = -1/1$, denoting whether a given neuron is silent/fires a spike in each temporal window. The network receives an external input, $\boldsymbol{x}$. The network state is described by a vector of n binary values, $\boldsymbol{\sigma} = \left(\sigma_1, \sigma_2, \ldots, \sigma_n\right)^T$. Both the network and input are assumed to have Markov dynamics. Neurons are updated asynchronously, by updating a random neuron at each time-step with probability $\pi_i\left(\sigma_i'|\boldsymbol{\sigma}\right)$. The network dynamics are thus described by:

$$p\left(\boldsymbol{\sigma}'|\boldsymbol{\sigma},\boldsymbol{x}\right) = \frac{1}{n}\sum_{i=1}^{n}\pi_i\left(\sigma_i'|\boldsymbol{\sigma},\boldsymbol{x}\right)\prod_{j\neq i}\delta(\sigma_j',\sigma_j). \tag{10}$$

where $\delta\left(\sigma_j',\sigma_j\right) = 1$ if $\sigma_j' = \sigma_j$ and 0 otherwise. Equivalently, we can say that at each time, a set of proposed updates, $\tilde{\boldsymbol{\sigma}}$, are independently sampled from $\tilde{\boldsymbol{\sigma}}_i \sim \prod_i \pi_i\left(\tilde{\sigma}_i|\boldsymbol{\sigma}\right)$, and then a neuron i is selected at random to be updated, such that $\sigma_i' \leftarrow \tilde{\sigma}_i$.

We define a reward function, $r(\boldsymbol{\sigma}, \boldsymbol{x})$, describing which states are 'desirable' for the network to perform a given function. The network dynamics are said to be optimal if they maximise the average reward, $\langle r(\boldsymbol{\sigma}, \boldsymbol{x})\rangle_{p(\boldsymbol{\sigma},\boldsymbol{x})}$ given a constraint on how much each neuron encodes about its inputs $\sum_i^n I(\tilde{\sigma}_i; \boldsymbol{\sigma}, \boldsymbol{x})$. This corresponds to maximising the objective function:

$$L = \langle r(\boldsymbol{\sigma}, \boldsymbol{x})\rangle_{p(\boldsymbol{\sigma},\boldsymbol{x})} - \lambda \sum_{i=1}^{n} I(\tilde{\sigma}_i; \boldsymbol{\sigma}, \boldsymbol{x}) \tag{11}$$

$$= \langle r(\boldsymbol{\sigma}, \boldsymbol{x}) - \lambda c(\boldsymbol{\sigma}, \boldsymbol{x})\rangle_{p(\boldsymbol{\sigma},\boldsymbol{x})} \tag{12}$$

where $c(\boldsymbol{\sigma}, \boldsymbol{x}) = \sum_{i=1}^{n} D_{KL}[\pi_i(\tilde{\sigma}_i|\boldsymbol{\sigma}, \boldsymbol{x}) \| p_i(\tilde{\sigma}_i)]$ is the coding cost associated with each state, and penalises deviations from each neuron's average firing rate.

We can decompose the transition probability for the network (Eqn 10), into the probability that a given neuron proposes an update, $\tilde{\sigma}_i$, given the network state, $\boldsymbol{\sigma}$, and the probability of the new network state, $\boldsymbol{\sigma}'$, given $\tilde{\sigma}_i$ and $\boldsymbol{\sigma}$:

$$p(\tilde{\sigma}_i|\boldsymbol{\sigma}) = \pi_i(\tilde{\sigma}_i|\boldsymbol{\sigma}) \tag{13}$$

$$p(\boldsymbol{\sigma}'|\boldsymbol{\sigma}, \tilde{\sigma}_i) = \frac{1}{n}\Big(\delta(\sigma_i', \tilde{\sigma}_i) \prod_{j\neq i} \delta(\sigma_j', \sigma_j) + \sum_{j\neq i} \pi_j(\sigma_j'|\boldsymbol{\sigma}) \prod_{k\neq j} \delta(\sigma_k', \sigma_k)\Big) \tag{14}$$

Thus, the problem faced by each neuron, of optimising $\pi_i(\tilde{\sigma}_i|\boldsymbol{\sigma})$ so as to maximise L_π, is equivalent to the MDP described in SI section 1, where the action a, and state s correspond to the neuron's proposed update $\tilde{\sigma}_i$ and state of the network and external inputs $\{\boldsymbol{\sigma}, \boldsymbol{x}\}$. It is thus straightforward to show that $\pi_i(\tilde{\sigma}_i|\boldsymbol{\sigma})$ can be optimised via the following updates:

$$v(\boldsymbol{\sigma}, \boldsymbol{x}) \leftarrow r(\boldsymbol{\sigma}, \boldsymbol{x}) - \lambda c(\boldsymbol{\sigma}, \boldsymbol{x}) + \langle v(\boldsymbol{\sigma}', \boldsymbol{x}')\rangle_{p(\boldsymbol{\sigma}',\boldsymbol{x}'|\boldsymbol{\sigma},\boldsymbol{x})} \tag{15}$$

$$\pi(\tilde{\sigma}_i|\boldsymbol{\sigma}, \boldsymbol{x}) \leftarrow \frac{p(\tilde{\sigma}_i)\, e^{\frac{1}{\lambda n}\langle v_\pi(\boldsymbol{\sigma}_{/i}, \tilde{\sigma}_i = \sigma_i, \boldsymbol{x}')\rangle_{p(\boldsymbol{x}'|\boldsymbol{x})}}}{Z(\boldsymbol{\sigma}_{/i}, \boldsymbol{x})} \tag{16}$$

where $Z(\boldsymbol{\sigma}_{/i}, \boldsymbol{x})$ is a normalisation constant, and $\boldsymbol{\sigma}_{/i}$ denotes the state of all neurons except neuron i. As updating the policy for any given neuron increases L_π, we can alternate updates for different neurons to optimise the dynamics of the whole network.

C. Inferring network function via inverse RL

After convergence, we can substitute the expression for the optimal policy into the Bellman equality, to obtain:

$$v(\boldsymbol{\sigma}, \boldsymbol{x}) = r(\boldsymbol{\sigma}, \boldsymbol{x}) + \lambda \sum_{i=1}^{n} \log \sum_{\sigma_i} p(\sigma_i)\, e^{\frac{1}{\lambda n}\langle v(\boldsymbol{\sigma}, \boldsymbol{x}')\rangle_{p(\boldsymbol{x}'|\boldsymbol{x})}} \tag{17}$$

Rearranging, we have:

$$r(\boldsymbol{\sigma}, \boldsymbol{x}) = v(\boldsymbol{\sigma}, \boldsymbol{x}) - \lambda \sum_{i=1}^{n} \log \sum_{\sigma_i} p(\sigma_i)\, e^{\frac{1}{\lambda n}\langle v(\boldsymbol{\sigma}, \boldsymbol{x}')\rangle_{p(\boldsymbol{x}'|\boldsymbol{x})}} \tag{18}$$

Thus, if we can infer the value function from the observed neural responses, then we can recover the associated reward function through Eqn 18.

First, we consider the case where there is no external input. In this case, the optimal neural dynamics (Eqn 16) correspond to Gibbs sampling from:

$$p(\boldsymbol{\sigma}) \propto \left(\prod_{i=1}^{n} p(\sigma_i)\right) e^{\frac{1}{\lambda n} v(\boldsymbol{\sigma})}. \tag{19}$$

Rearranging, we have a closed-form expression for the value function, $v(\boldsymbol{\sigma})$, in terms of the steady-state distribution:

$$v(\boldsymbol{\sigma}) = n\lambda \log\left(\frac{p(\boldsymbol{\sigma})}{\prod_{i=1}^{n} p(\sigma_i)}\right) + C \tag{20}$$

where C is an irrelevant constant. We can then combine Eqn 20 and Eqn 18 to obtain a closed-form expression for the reward function:

$$r(\boldsymbol{\sigma}) = n\lambda \sum_{i=1}^{n} \log\left(\frac{p(\sigma_i|\boldsymbol{\sigma}_{/i})}{p(\sigma_i)}\right) + C. \tag{21}$$

Since we don't know the true value of λ, we can simply set it to unity. In this case, our inferred reward will differ from the true reward by a factor of $\frac{1}{\lambda}$. However, since dividing both the reward and coding cost by the same factor has no effect on the shape of the objective function, L_π (only its magnitude), this will not effect any predictions we make using the inferred reward.

With an external input, there is no closed-form solution for the value function. Instead, we can infer $v(\boldsymbol{\sigma}, \boldsymbol{x})$ numerically, by maximising the log-likelihood of the optimal network dynamics, $\langle \log p^*(\boldsymbol{\sigma}'|\boldsymbol{\sigma}, \boldsymbol{x})\rangle_{\mathcal{D}} = \left\langle \log \sum_{i=1}^{n} \pi_i^*(\sigma_i'|\boldsymbol{\sigma}, \boldsymbol{x})\, \delta\left(\boldsymbol{\sigma}'_{/i}, \boldsymbol{\sigma}_{/i}\right)\right\rangle_{\mathcal{D}}$, where $\pi_i^*(\sigma_i'|\boldsymbol{\sigma}, \boldsymbol{x})$ denote optimal transition probabilities.

D. Approximate method for larger networks

1. RL

To scale our framework to larger networks we approximate the value function, $v(\boldsymbol{\sigma}, \boldsymbol{x})$, by a parametric function of the network activity, $\boldsymbol{\sigma}$ and input, $\boldsymbol{x}$. Without loss of generality, we can parameterise the value function as a linear combination of basis functions: $v_{\boldsymbol{\phi}}(\boldsymbol{\sigma}, \boldsymbol{x}) \equiv \boldsymbol{\phi}^T \boldsymbol{f}(\boldsymbol{\sigma}, \boldsymbol{x})$. From Eqn 17, if the network is optimal, then the value function equals:

$$\hat{v}_\phi(\boldsymbol{\sigma}, \boldsymbol{x}) = r(\boldsymbol{\sigma}, \boldsymbol{x}) + \lambda \sum_{i=1}^{n} \log \sum_{\sigma_i} p(\sigma_i)\, e^{\frac{1}{n\lambda}\boldsymbol{\phi}^T \boldsymbol{f}(\boldsymbol{\sigma}, \boldsymbol{x})}. \tag{22}$$

In the exact algorithm, we updated the value function by setting it equal to $\hat{v}(\boldsymbol{\sigma}, \boldsymbol{x})$ (Eqn 15). Since in the parametric case this is not possible, we can instead update $\boldsymbol{\phi}$ to minimise:

$$G_{\bar{\boldsymbol{\phi}}}(\boldsymbol{\phi}) = \frac{1}{2}\left\langle \left(v_{\boldsymbol{\phi}}(\boldsymbol{\sigma}, \boldsymbol{x}) - \hat{v}_{\bar{\boldsymbol{\phi}}}(\boldsymbol{\sigma}, \boldsymbol{x})\right)^2 \right\rangle_{p(\boldsymbol{\sigma}, \boldsymbol{x})}, \tag{23}$$

where $\hat{v}_{\bar{\boldsymbol{\phi}}}(\boldsymbol{\sigma}, \boldsymbol{x})$ is the target value function, defined as in Eqn 22, with parameters, $\bar{\boldsymbol{\phi}}$ [3–5].

We follow the procedure set out in [3–5] to transform this into a stochastic gradient descent algorithm. First, we perform n_{batch} samples from the current policy. Next, we perform a stochastic gradient descent update:

$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \frac{\eta}{n_{batch}} \sum_{l=1}^{n_{batch}} \boldsymbol{f}(\boldsymbol{\sigma}_l, \boldsymbol{x}_l)\left(v_{\boldsymbol{\phi}}(\boldsymbol{\sigma}_l, \boldsymbol{x}_l) - \hat{v}_{\bar{\boldsymbol{\phi}}}(\boldsymbol{\sigma}_l, \boldsymbol{x}_l)\right) \tag{24}$$

where η is a constant that determines the learning rate. Finally, after doing this n_{epoch} times, we update the target parameters, $\bar{\boldsymbol{\phi}} \leftarrow \boldsymbol{\phi}$. These steps are repeated until convergence.

2. *Inverse RL*

We can infer the parameters of the value function, $\boldsymbol{\phi}$, by maximising the log-likelihood: $\langle \log p_{\boldsymbol{\phi}}(\boldsymbol{\sigma}'|\boldsymbol{\sigma}, \boldsymbol{x})\rangle_{\mathcal{D}}$. We can choose the form of the value function to ensure that this is tractable. For example, if the value function is quadratic in the responses, then this corresponds to inferring the parameters of a pair-wise Ising model [6, 7].

After inferring $\boldsymbol{\phi}$, we want to infer the reward function. At convergence, $\nabla_{\boldsymbol{\phi}} F(\boldsymbol{\phi}) = \mathbf{0}$ and $\bar{\boldsymbol{\phi}} = \boldsymbol{\phi}$, so that:

$$\mathbf{0} = \langle \boldsymbol{f}(\boldsymbol{\sigma}, \boldsymbol{x})\left(v_{\boldsymbol{\phi}}(\boldsymbol{\sigma}, \boldsymbol{x}) - \hat{v}_{\boldsymbol{\phi}}(\boldsymbol{\sigma}, \boldsymbol{x})\right)\rangle_{\mathcal{D}} \tag{25}$$

$$= \langle \boldsymbol{f}(\boldsymbol{\sigma}, \boldsymbol{x})\left(r(\boldsymbol{\sigma}, \boldsymbol{x}) - \hat{r}_{\boldsymbol{\phi}}(\boldsymbol{\sigma}, \boldsymbol{x})\right)\rangle_{\mathcal{D}} \tag{26}$$

where $\hat{r}(\boldsymbol{\sigma}, \boldsymbol{x})$ is given by:

$$\hat{r}_{\boldsymbol{\phi}}(\boldsymbol{\sigma}, \boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{\sigma}, \boldsymbol{x})^T \boldsymbol{\phi} - \lambda \sum_{i=1}^{n} \log \sum_{\sigma_i} p(\sigma_i)\, e^{\frac{1}{n\lambda}\boldsymbol{\phi}^T \boldsymbol{f}(\boldsymbol{\sigma}, \boldsymbol{x})} \tag{27}$$

In the exact case, where $v_{\boldsymbol{\phi}}(\boldsymbol{\sigma}, \boldsymbol{x}) = \hat{v}_{\boldsymbol{\phi}}(\boldsymbol{\sigma}, \boldsymbol{x})$ (and thus, $G_{\bar{\boldsymbol{\phi}}}(\boldsymbol{\phi}) = 0$), then the inferred reward equals $\hat{r}_{\boldsymbol{\phi}}(\boldsymbol{\sigma}, \boldsymbol{x})$. However, this is not necessarily true when we assume an approximate value function.

Just as we did for the value function, we can express the reward function as a linear combination of basis functions: $r(\boldsymbol{\sigma}, \boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{g}(\boldsymbol{\sigma}, \boldsymbol{x})$. Thus, Eqn 26 becomes:

$$\left\langle \boldsymbol{f}(\boldsymbol{\sigma})\, \boldsymbol{g}(\boldsymbol{\sigma})^T \right\rangle_{\mathcal{D}} \boldsymbol{\theta} = \langle \boldsymbol{f}(\boldsymbol{\sigma}, \boldsymbol{x})\, \hat{r}_{\boldsymbol{\phi}}(\boldsymbol{\sigma}, \boldsymbol{x})\rangle_{\mathcal{D}}. \tag{28}$$

If the reward function has the same number of parameters than the approximate value function (i.e. $\boldsymbol{f}(\boldsymbol{\sigma})$ and $\boldsymbol{g}(\boldsymbol{\sigma})$ have the same size), then we can solve this equation to find $\boldsymbol{\theta}$. Alternatively, if the reward function has more parameters than the value function, then we require additional assumptions to unambiguously infer the reward.

II. SIMULATION DETAILS

A. Agent navigating a maze

We considered an agent navigating a 15×15 maze. The agent could choose to move up, down, left or right in the maze. At each time there was a 5% probability that the agent moved in a random direction, independent of their selected action. Moving in the direction of a barrier (shown in blue in Fig 1D) would result in the agent remaining in the same location. After reaching the 'rewarded' location (bottom right of the maze), the agent was immediately transported to a starting location in the top left of the maze. We optimised the agent's policy at both low and high coding cost ($\lambda = 0.013/0.13$ respectively). The reward was inferred from the agent's policy after optimisation as described in SI section IA.1.

B. Network with single binary input

We simulated a network of 8 neurons that receive a single binary input, x. The stimulus has a transition probability: $p(x' = 1|x = -1) = p(x' = -1|x = 1) = 0.02$. The reward function was unity when $x = 1$ and the network fired exactly 2 spikes, or when $x = -1$ and the network fired exactly 6 spikes. We set $\lambda = 0.114$.

To avoid trivial solutions where a subset neurons spike continuously while other neurons are silent, we defined the coding cost to penalise deviations from the *population averaged* firing rate (rather than the average firing rate for each neuron). Thus, the coding cost was defined as $c(\boldsymbol{\sigma}) = \sum_{i=1}^{n} \langle \log \pi_i(\tilde{\sigma}_i|\boldsymbol{\sigma})/p(\tilde{\sigma}_i)\rangle_{\pi_i(\tilde{\sigma}_i|\boldsymbol{\sigma})}$, where $p(\tilde{\sigma}_i)$ is the average spiking probability, across all neurons.

We inferred the reward $r(\boldsymbol{\sigma})$ from neural responses as described in SI section 1C. In Fig 2E we rescaled and shifted the inferred reward to have the same mean and variance as the true reward.

We used the inferred reward to predict how neural tuning curves should adapt when we alter the stimulus statistics (Fig 2F, upper) or remove a cell (Fig 2F, lower). For fig 2F (upper), we altered the stimulus statistics by setting $p(x' = 1|x = -1) = 0.01$ and $p(x' = -1|x = 1) = 0.03$. For figure 2C (lower), we removed one cell from the network. In both cases, we manually adjusted λ to keep the average coding cost constant.

C. Pair-wise coupled network

We considered a network of 12 neurons arranged in a ring. We defined a reward function that was equal to 1 if exactly 4 adjacent neurons were active, and 0 otherwise. We defined the coding cost as described in the previous section, to penalise deviations from the population averaged mean firing rate.

We approximated the value function by a quadratic

function, $v(\boldsymbol{\sigma}) = \sum_{i,j\neq i} J_{ij}\sigma_i\sigma_j + \sum_i h_i\sigma_i$. We optimised the parameters of this value function using the algorithm described in SI section 1D, with $\lambda = 0.05$. We used batches of $n_{batch} = 40$ samples, and updated the target parameters after every $n_{epoch} = 100$ batches.

We inferred the reward function from the inferred network couplings, $\boldsymbol{J}$ and $\boldsymbol{h}$. As described in SI section 1D, this problem is only well-posed if we assume a low-d parametric form for the reward function, or add additional assumptions. For our simulation, we assumed the reward function was always positive ($r(\boldsymbol{\sigma}) > 0$) and sparse (i.e. we penalised large values of $l_1 = \sum_{\boldsymbol{\sigma}} r(\boldsymbol{\sigma})$).

Finally, for the simulations shown in panels 3B-D, we ran the optimisation with $\lambda = 0.1, 0.01$ and 0.1, respectively. For panel 3C we removed connections between neurons separated by a distance of 3 or more on the ring. For panel 3D we forced two of the neurons to be continuously active.

D. Efficient coding

We considered a stimulus consisting of $m = 7$ binary variables, $x_i = -1/1$. The stimulus had Markov dynamics, with each unit updated asynchronously. The stimulus dynamics were given by:

$$p(\boldsymbol{x}'|\boldsymbol{x}) = \frac{1}{m}\sum_{i=1}^{m}\left(1 + e^{J\sum_{j=1}^{m} x_j}\right)^{-1}\delta\left(\boldsymbol{x}'_{/i}, \boldsymbol{x}\right), \quad (29)$$

where $J = 1.5$ is a coupling constant. A 'relevant' variable, $y(\boldsymbol{x})$ was equal to 1 if 4 or more inputs equalled 1, and equal to -1 otherwise.

We optimised a network of $n = 7$ neurons to efficiently code the relevant variable $y(\boldsymbol{x})$, using the algorithm described in the main text. For Fig 4B-C we set $\lambda = 0.167$. For Fig 4D we varied λ between 0.1 and 0.5. For Fig 4E we altered the stimulus statistics so that,

$$p(\boldsymbol{x}'|\boldsymbol{x}) = \frac{1}{m}\sum_{i=1}^{m}\left(1 + e^{J_0 + J\sum_{j=1}^{m} x_j}\right)^{-1}\delta\left(\boldsymbol{x}'_{/i}, \boldsymbol{x}\right), \quad (30)$$

where J_0 was a bias term that we varied between 0 and 0.4. For each value of J_0 we adjust λ so as to keep the average coding cost constant.

[1] Sutton RS, Barto AG (2018). Reinforcement learning: An introduction. MIT press.

[2] Mahadevan S (1996) Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine learning*, 22:159-195.

[3] Schulman J, Chen X, Abbeel P (2017) Equivalence between policy gradients and soft Q-learning. arXiv:1704.06440.

[4] Haarnoja T, Tang H, Abbeel P, Levine S (2017). Reinforcement learning with deep energy-based policies. *Proc 34th International Conf on Machine Learning* 70:1352-1361

[5] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumuran D, Wierstra D, Legg S, Hassabis D (2015) Human-level control through deep reinforcement learning. *Nature*, 518:529–533

[6] Schneidman E, Berry II MJ, Segev R, Bialek W. Weak pairwise correlations imply strongly correlated network states in a neural population. Nature 440:1007–1012.

[7] Tkačik G, Marre O, Amodei D, Schneidman E, Bialek W, Berry II MJ. (2014) Searching for collective behavior in a large network of sensory neurons. *PLoS Comp Bio*, 10:e1003408.