Inferring Neural Activity Before Plasticity: A Foundation for Learning Beyond Backpropagation

Yuhang Song[1,2,*], **Beren Millidge**[2], **Tommaso Salvatori**[1], **Thomas Lukasiewicz**[1,*], **Zhenghua Xu**[1,3], **and Rafal Bogacz**[2,*]

[1]Department of Computer Science, University of Oxford, Oxford, United Kingdom
[2]Medical Research Council Brain Networks Dynamics Unit, University of Oxford, Oxford, United Kingdom
[3]State Key Laboratory of Reliability and Intelligence of Electrical Equipment, Hebei University of Technology, Tianjin, China
[*]Corresponding authors: yuhang.song@bndu.ox.ac.uk; thomas.lukasiewicz@cs.ox.ac.uk; rafal.bogacz@ndcn.ox.ac.uk

Abstract

For both humans and machines, the essence of learning is to pinpoint which components in its information processing pipeline are responsible for an error in its output — a challenge that is known as *credit assignment*[1]. How the brain solves credit assignment is a key question in neuroscience, and also of significant importance for artificial intelligence. Many recent studies[1–12] presuppose that it is solved by backpropagation[13–16], which is also the foundation of modern machine learning[17–22]. However, it has been questioned whether it is possible for the brain to implement backpropagation[23,24], and learning in the brain may actually be more efficient and effective than backpropagation[25]. Here, we set out a fundamentally different principle on credit assignment, called *prospective configuration*. In prospective configuration, the network first infers the pattern of neural activity that should result from learning, and then the synaptic weights are modified to consolidate the change in neural activity. We demonstrate that this distinct mechanism, in contrast to backpropagation, (1) underlies learning in a well-established family of models of cortical circuits, (2) enables learning that is more efficient and effective in many contexts faced by biological organisms, and (3) reproduces surprising patterns of neural activity and behaviour observed in diverse human and animal learning experiments. Our findings establish a new foundation for learning beyond backpropagation, for both understanding biological learning and building artificial intelligence.

The credit assignment problem[1] lies at the very heart of learning. *Backpropagation*[13–16], as a simple yet effective credit assignment theory, has powered notable advances in artificial intelligence since its inception[17–22]. It has also gained a predominant place in understanding learning in the brain[1,2,4–9,11,12,26]. Due to this success, much recent work has focused on understanding how biological neural networks could learn in a way similar to backpropagation[27–36]: although many proposed models do not implement backpropagation exactly, they nevertheless try to approximate backpropagation, and much emphasis is placed on how close this approximation is[3,27–33,37,38]. However, learning in the brain is superior to backpropagation in many critical aspects — for example, compared to the brain, backpropagation requires many more exposures to a stimulus to learn[25] and suffers from catastrophic interference of newly and previously stored information[39,40]. This raises the question of whether using backpropagation to understand learning in the brain should be the main focus of the field.

Here, we propose that the brain instead solves credit assignment with a fundamentally different principle, which we call *prospective configuration*. In prospective configuration, before synaptic weights are modified, neural activity changes across the network so that output neurons better predict the target output; only then are the synaptic weights (weights, for short) modified to consolidate this change in neural activity. By contrast, in backpropagation the order is reversed — weight modification takes the lead

and the change in neural activity is the result that follows.

We identify prospective configuration as a principle that is implicitly followed by a well-established family of neural models with solid biological groundings, namely, energy-based networks. They include Hopfield networks[41] and predictive coding networks[42], which have been successfully used to describe information processing in the cortex[43–49]. To support the theory of prospective configuration, we show that it can both yield efficient learning, which humans and animals are capable of, and reproduce data from experiments on human and animal learning. Thus, on one hand, we demonstrate that prospective configuration performs more efficient and effective learning than backpropagation in various situations faced by biological systems, such as learning with deep structures, online learning, learning with a limited amount of training examples, learning in changing environments, continual learning with multiple tasks, and reinforcement learning. On the other hand, we demonstrate that patterns of neural activity and behaviour in diverse human and animal learning experiments, including sensorimotor learning, fear conditioning and reinforcement learning, can be naturally explained by prospective configuration, but not by backpropagation.

Guided by the belief that backpropagation is the foundation of biological learning, previous work showed that energy-based networks can closely approximate backpropagation. However, to achieve it, the networks were set up in an unnatural way, such that the neural activity was prevented from substantially changing before weight modification, by constraining the supervision signal to be infinitely small or last an infinitely short time[29,30,37,50,51]. In contrast, we reveal that the energy-based networks without these unrealistic constrains follow the distinct principle of prospective configuration rather than backpropagation, and are superior in both learning efficiency and accounting for data on biological learning.

Below, we first introduce prospective configuration with an intuitive example, show how it originates from energy-based networks, describe its advantages and quantify them in a rich set of biological-relevant learning tasks. Finally, we show that it naturally explains patterns of neural activity and behaviour in diverse learning experiments.

Results

Prospective configuration: an intuitive example

To optimally plan behaviour, it is critical for the brain to predict future stimuli — for example, to predict sensations in some modalities on the basis of other modalities[52]. If the observed outcome differs from the prediction, the weights in the whole network need to be updated so that prediction in the "output" neurons are corrected. Backpropagation computes how the weights should be modified to minimize the error on the output, and this weight update results in the change of neural activity when the network next makes the prediction. In contrast, we propose that the activity of neurons is first adjusted to a new configuration, so that the output neurons better predict the observed outcome (target pattern); the weights are then modified to reinforce this configuration of neural activity. We call this configuration of neural activity "prospective", since it is the neural activity that the network *should produce* to correctly predict the observed outcome. In agreement with the proposed mechanism of prospective configuration, it has indeed been widely observed in biological neurons that presenting outcome of a prediction triggers changes in neural activity — for example, in tasks requiring animals to predict a fruit juice delivery, the reward triggers rapid changes in activity not only in the gustatory cortex, but also in multiple cortical regions[53,54].

To highlight the difference between backpropagation and prospective configuration, consider a simple example in Fig. 1a. Imagine a bear seeing a river. In the bear's mind, the sight generates predictions of hearing water and smelling salmon. On that day, the bear indeed smelled the salmon but did not hear the water, perhaps due to an ear injury, and thus the bear needs to change its expectation related

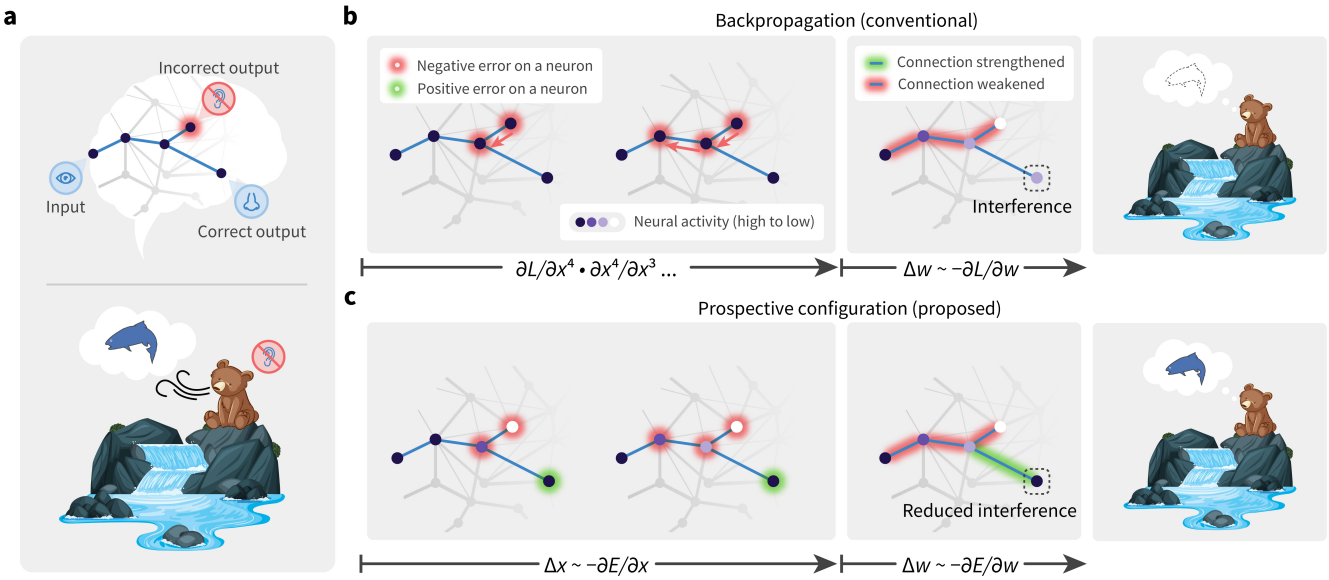


Fig. 1| Prospective configuration avoids interference during learning. ►**a**| An abstract (top) and a concrete (bottom) example of a task inducing interference during learning. One stimulus input (seeing the water) triggers two prediction outputs (hearing the water and smelling the salmon). One output is correct (smelling the salmon), while the other one is an error (not hearing the water). Backpropagation produces interference during learning: not hearing the water reduces the expectation of smelling the salmon (panel b), although the salmon was indeed smelled. Prospective configuration, on the other hand, avoids such interference (panel c). ►**b**| In backpropagation, negative error propagates from the error output into hidden neurons (left). This causes a weakening of some connections, which on the next trial improves the incorrect output, but it also reduces the prediction of the correct output, thus introducing interference (middle and right). ►**c**| In prospective configuration, neural activity settles into a new configuration (purple of different intensity) before weight modification (left). This configuration corresponds to the activity that should be produced after learning, i.e., is "prospective". Hence it foresees the positive error on the correct output, and modifies the connections to improve the incorrect output, while maintaining the correct output (middle and right).

to the sound. Backpropagation (Fig. 1b) would proceed by backpropagating the negative error, so as to reduce the weights on the path between the visual and auditory neurons. However, this modification would also reduce the expectation of smelling the salmon the next time the river is visited, even though the smell of salmon was present and correctly predicted. These undesired and unrealistic side effects of learning with backpropagation are closely related with the phenomenon of catastrophic interference, where learning a new association destroys previously learned memories[39,40]. This example shows that, with backpropagation, even learning one new aspect of an association may interfere with the memory of other aspects of the same association.

In contrast, prospective configuration assumes that learning starts with the neurons being configured to a new state — which corresponds to a pattern enabling the network to correctly predict the observed outcome. The weights are then modified to consolidate this state. This behaviour can "foresee" side effects of potential weight modifications and compensate for them dynamically — Fig. 1c: to correct the negative error on the incorrect output, the hidden neurons settle to their prospective state of lower activity, and as a result, a positive error is revealed and allocated to the correct output. Consequently, prospective configuration increases the weights connecting to the correct output, while backpropagation does not (cf. middle plots of Fig. 1b and c). Hence, prospective configuration is able to correct the side effects of learning an association effectively, efficiently, and with little interference.

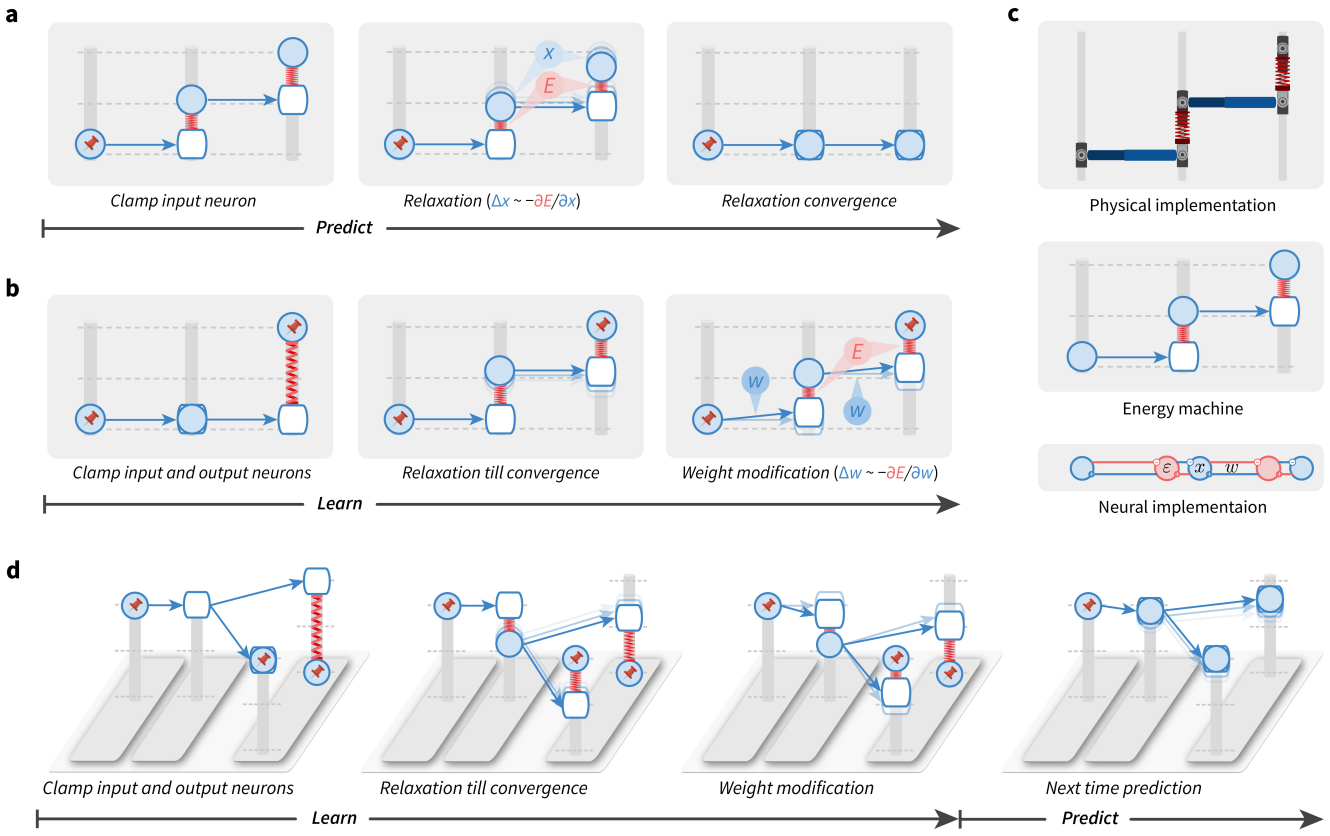


Fig. 2| The energy machine reveals a new understanding of energy-based networks, the mechanism of prospective configuration, and its theoretical advantages. A subset of energy-based networks can be visualized as mechanical machines that perform equivalent computations. Here, we present one of them, predictive coding networks[30,43,55]. In the energy machine, the activity of a neuron corresponds to a height of a node (represented by a solid circle) sliding on a post. The input to the neuron is represented by a hollow node on the same post. A synaptic connection corresponds to a rod pointing from a solid to a hollow node. The synaptic weight determines how the input to a post-synaptic neuron depends on the activity of pre-synaptic neuron, hence it influences the angle of the rod. In these energy-based networks, the activity of a neuron is not fixed to its input, instead, an energy function is defined, which is equivalent to an elastic potential energy of springs connecting nodes on the posts (the natural length of the springs is zero). In energy-based networks, relaxation (i.e., neural dynamics) and weight modification (i.e., weight dynamics) are both driven by minimizing the energy, thus correspond to relaxing the energy machine by moving the nodes and tuning the rods, respectively. ►**a-b**| Predictions (a) and learning (b) in energy-based networks, visualized by the energy machine. The pin indicates that the neural activity is fixed to the input or target pattern. Here, it is revealed that the relaxation infers the prospective neural activity, towards which the weights are then modified, a mechanism that we call prospective configuration. ►**c**| The physical implementation (top) and the connectivity of a predictive coding network[30,43,55] (bottom), which has a dynamics mathematically equivalent to the energy machine in the middle (see Methods for details). ►**d**| The learning problem in Fig. 1, visualized by the energy machine, which learns to improve the incorrect output while not interfering with the correct output, thanks to the mechanism of prospective configuration.

Origin of prospective configuration: energy-based networks

To shows how prospective configuration naturally arises in energy-based networks, we introduce a physical machine analog, that provides an intuitive understanding of energy-based networks, and how they produce the mechanism of prospective configuration.

Energy-based networks have been widely and successfully used in describing biological neural

systems[41,42,56–58]. In these models, a neural circuit is described by a dynamical system driven by reducing an abstract "energy", e.g., reflecting errors made by the neurons; see Methods. Neural activity and synaptic weights change to reduce this energy, hence they can be considered as "movable parts" of the dynamical system. We show below that energy-based networks are mathematically equivalent to a physical machine (we call it *energy machine*), where the energy function has an intuitive interpretation and its dynamics are straightforward — the energy machine simply adjusts its movable parts to reduce energy.

As shown in Fig. 2a–b, the energy machine includes nodes sliding on vertical posts, connected with each other via rods and springs. Translating from energy-based networks to the energy machine, the neural activity maps to the vertical position of a solid node, a connection maps to a rod (blue arrow) pointing from one node to another (where the weight determines how the end position of the rod relates to the initial position), and the energy function maps to the elastic potential energy of springs with nodes attached on their both ends (the natural length of the springs is zero). Different energy functions and networks structures result in different energy-based networks, corresponding to energy machines with different configurations and combinations of nodes, rods, and springs. In Fig. 2, we present the energy machine of predictive coding networks[30,43,55], because they are most accessible and established to be closely related to backpropagation[30,37].

The dynamics of energy-based networks, which are driven by minimizing the energy function, maps to the relaxation of the energy machine, which is driven by reducing the total elastic potential energy on the springs. A prediction with energy-based networks involves clamping the input neurons to the provided stimulus and updating the activity of the other neurons, which corresponds to fixing one side of the energy machine and letting the energy machine relax by moving nodes (Fig. 2a). Learning with energy-based networks involves clamping the input and output neurons to the corresponding stimulus, first letting the activity of the remaining neurons converge and then updating weights, which corresponds to fixing both sides of the energy machine and letting the energy machine relax first by moving nodes and then by tuning rods (Fig. 2b).

The energy machine reveals the essence of energy-based networks: the relaxation before weight modification lets the network settle to a new configuration of neural activity, corresponding to those that would have occurred after the error was corrected by the modification of weights, i.e., prospective activity (thus, we call this mechanism prospective configuration). For example, the second layer "neuron" in Fig. 2b increases its activity, and this increase in activity would also be caused by the subsequent weight modification (of the connection between the first and the second neurons). In simple terms, the relaxation in energy-based networks infers the prospective neural activity after learning, towards which the weights are then modified. This distinguishes it from backpropagation, where the weights modification takes the lead, and the change in neural activity is the result that follows.

The bottom part of Fig. 2c shows the connectivity of a predictive coding network[30,43,55], which has a dynamics mathematically equivalent to the energy machine shown above it. Predictive coding networks include neurons (blue) corresponding to nodes on the posts, and separate neurons encoding prediction errors (red) corresponding to springs, (see Methods for details).

Using the energy machine, Fig. 2d simulates the learning problem from Fig. 1. Here, we can see that prospective configuration indeed foresees the result of learning and its side effects, through relaxation. Hence, it learns to avoid interference within one iteration, which would otherwise take multiple iterations for backpropagation.

Advantages of prospective configuration: reduced interference and faster learning

Here we quantify interference in the above scenario and demonstrate how the reduced interference translates into an advantage in performance. In all simulations in the main text prospective configuration

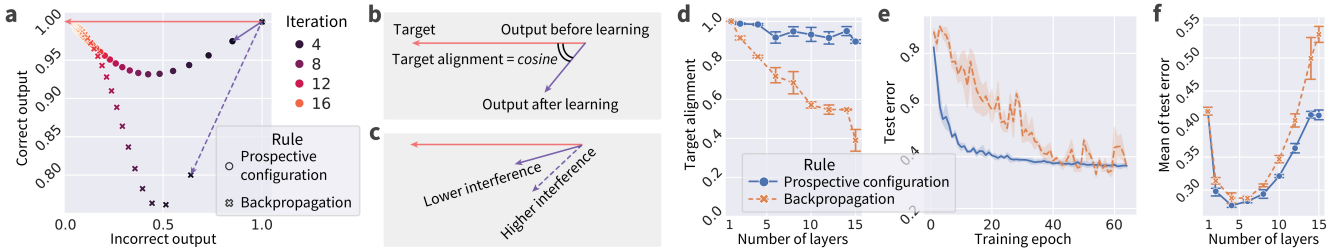


Fig. 3| Learning with prospective configuration changes the activity of output neurons in a direction more aligned towards the target. ► **a**| Simulation of network from Fig. 1 showing changes of the correct and incorrect output neurons during training ("Iteration"), trained with both learning rules. Here, learning with prospective configuration (purple solid vector) aligns better with the target (red vector), than for backpropagation (purple dashed vector). ► **b**| The interference can be quantified by "target alignment": the cosine similarity of the direction of target (red vector) and the direction of learning (purple vector). ► **c**| Higher target alignment indicates less interference and vice versa. ► **d**| Target alignment of randomly generated networks trained with both learning rules, as a function of depth of the network. Here, target alignment drops as the network gets deeper, demonstrating the difficulty of training deep structures. However, prospective configuration maintains much higher target alignment along the way. ► **e**| Classification error during training on FashionMNIST dataset containing images of clothing belonging to different categories, for both learning rules, with a deep neural network of 15 layers. ► **f**| Mean of classification error plot in panel e (reflecting how fast test error drops) and of other network depths. In panels e-f, prospective configuration demonstrates notable advantage as the structure gets deep.

is implemented in predictive coding networks (see Methods). Fig. 3a compares the activity of output neurons in the example in Fig. 1, between backpropation and prospective configuration. Initially both output neurons are active (top right corner), and the output should change towards a target in which one of the neurons is inactive (red vector). Learning with prospective configuration results in changes on the output (purple solid vector) that are aligned better with the target than those for backpropagation (purple dotted vector). Although the output from backpropagation can reach the target after multiple iterations, the output for the "correct neuron" diverges from the target during learning and then comes back - it is particularly undesired effect in biological learning, where networks can be "tested" at any point during the learning process, because it may lead to incorrect decisions affecting chances for survival. By contrast, prospective configuration substantially reduces this effect.

The interference can be quantified by the angle between the direction of target and learning, and we define "target alignment" as the cosine of this angle (Fig. 3b), hence high interference corresponds to low target alignment (Fig. 3c). The difference in target alignment demonstrated in Fig. 3a is also present for deeper and larger (randomly generated) networks, as shown in Fig. 3d. When a network has no hidden layers, the target alignment is equal to 1 (proved in section 2.2 of Supplementary Information). The target alignment drops for backpropagation as the network gets deep, because changes in weights in one layer may interfere with changes in other layers (as explained in Fig. 1); while prospective configuration maintains a much higher value along the way. This metric directly translates to the efficiency of learning: Fig. 3e shows that the test error during training in a visual classification task with a deep neural network of 15 layers decreases faster for prospective configuration than backpropagation. Fig. 3f repeats the experiment on networks of other depths, and shows the mean of the test error during training (reflecting how fast the test error drops), as a function of network depth. The mean error is higher for low depths, as these networks are unable to learn the task, and for greater depths, because it takes longer to train deeper networks. Importantly, the gap between backpropagation and prospective configuration widens for deeper networks, paralleling the difference in target alignment. Efficient training with deeper networks is important for biological neural systems, known to be deep, e.g., primate visual cortex[59].

In the Supplementary Information we develop a formal theory of prospective configuration and provide further illustrations and analyses of its advantages. Extended Data Fig. 1 formally defines prospective configuration and demonstrates that it is indeed commonly observed in different energy-based networks. Extended Data Fig. 2 shows that prospective configuration shares a close relationship with another influential model of credit assignment called target propagation[60], providing another way of understanding the advantages of prospective configuration. Extended Data Figs. 3 and 4 empirically verify and generalize the advantages expected from the theory: they show that prospective configuration yields more accurate error allocation and less erratic weight modification, respectively.

Advantages of prospective configuration: effective learning in biologically relevant scenarios

Inspired by these advantages, we show empirically that prospective configuration indeed handles various learning problems that biological systems would face better than backpropagation. Since the field of machine learning has developed effective benchmarks for testing learning performance, we use variants of classic machine learning problems that share key features with the learning in natural environments. Such problems include online learning where the weights must be updated after each experience (rather than a batch of training examples)[61], learning with limited amount of training examples, learning in changing environments[62], continual learning with multiple tasks[63,64], and reinforcement learning[21]. In all the aforementioned learning problems, prospective configuration demonstrates a notable superiority over backpropagation.

Firstly, based on the example in Fig. 1, we expect prospective configuration to require fewer episodes for learning than backpropagation. Before presenting the comparison, we describe how backpropagation is used to train artificial neural networks. Typically, the weights are only modified after a batch of training examples, based on the average of updates derived from individual examples (Fig. 4a). In fact, backpropagation relies heavily on averaging over multiple experiences to reach human-level performance[67–69] as it needs to stabilise training[70]. By contrast, biological systems must update the weights after each experience, and we compare the learning performance in such a setting. The sampling efficiency can be quantified by mean of test error during training, which is shown in Fig. 4b as a function of batch size (number of experiences that the updates are averaged over). The efficiency strongly depends on batch size for backpropagation, because it requires batch-training to average out erratic weight updates, while this dependence is weaker for prospective configuration, where the weight changes are intrinsically less erratic and the batch-averaging is less required (see Extended Data Fig. 4). Importantly, prospective configuration learns faster with smaller batch sizes, as in biological settings. Additionally, the final performance can be quantified by the minimum of the test error, which is shown in Fig. 4c, for both the FashionMNIST and the CIFAR-10[66] natural image datasets, when trained with batch size equals to one. Here, prospective configuration also demonstrates a notable advantage over backpropagation.

Secondly, biological learning is also characterized by a limited data availability. Fig. 4d show that prospective configuration outperforms backpropagation when the model is trained with fewer examples.

Thirdly, biological systems often need to rapidly adapt to changing environments. A common way to simulate this is "concept drifting"[62], where a part of the mapping between the output neurons to the semantic meaning is shuffled every period of time (Fig. 4e left). Fig. 4e right shows the test error during training with concept drifting. The advantage of prospective configuration demonstrated here is related to it being able to optimally detect which weights to modify (see Extended Data Fig. 3). Fig. 4f summarizes the results and repeat the experiment on CIFAR-10. Here, we can see that prospective configuration learns better with concept drifting, indicating a better adaptation to changing environments.

Furthermore, biological organisms need to sequentially learn multiple tasks, while artificial neural networks show catastrophic forgetting: when trained on a new task, performance on previously learnt

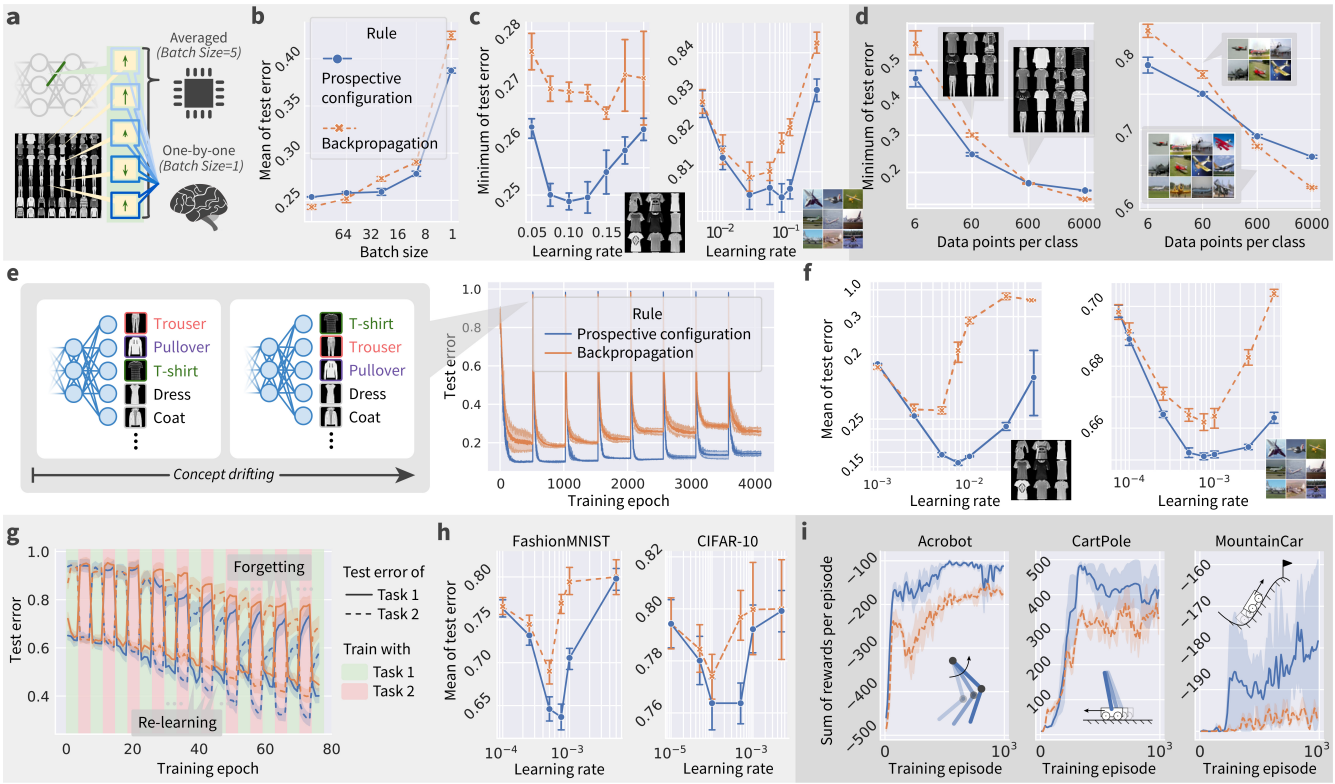


Fig. 4| Prospective configuration achieves a superior performance over backpropagation in various learning situations faced by biological systems. These situations are: online learning[61] (a–d), learning with a limited amount of training examples and limited size of architecture (e–f), learning in changing environments[62] (g–h), continual learning of multiple tasks[63,64] (i–j), reinforcement learning[21] (k). Every supervised learning setup (a-j) is evaluated on both FashionMNIST[65] (default) and CIFAR-10[66] datasets. If the learning rate is not presented in a plot, the plot corresponds to the best learning rate for each rule. ►**a**| Difference in training setup between computers that can average weight modifications for individual examples to get a "statistically good" value, and biological systems which must apply one modification before computing another. ►**b**| Mean of the test errors during training, as a function of batch size. ►**c**| Minimum of the test error during training as a function of learning rate on both datasets. ►**d**| Minimum of the test errors during training, with different amounts of training examples (datapoints per class) on both datasets. ►**e**| Test error during training when learning with concept drifting. ►**f**| Mean of test error during training with concept drifting as a function of learning rate. ►**g**| Test error during continual learning of two tasks. ►**h**| Mean of test error of both tasks during training as a function of learning rate. ►**i**| Sum of rewards per episode during training on three classic reinforcement learning tasks (insets). An episode is a period from initialization of environment to reaching a terminate state.

tasks is largely destroyed[39,71–73]. Fig. 4g shows the performance when trained on two tasks alternately (task 1 is classifying five randomly selected classes in FashionMNIST dataset, and task 2 is classifying the remaining five classes). It shows that prospective configuration outperforms backpropagation in both terms of avoiding forgetting previous tasks and re-learning current tasks. Fig. 4h summarizes the results and repeat the experiment on CIFAR-10.

Another key challenge for biological systems is to decide which actions to take. Reinforcement learning theories (e.g., *Q*-learning) propose that it is solved by learning the expected reward resulting from different actions in different situations[74]. Such prediction of rewards can be made by neural networks[21], which can be trained with prospective configuration or backpropagation. The sum of rewards per episode during training on three classic reinforcement learning tasks is reported in Fig. 4i, where prospective

configuration demonstrates a notable advance over backpropagation. This large advantage may arise because reinforcement learning is particularly sensitive to erratic changes in network's weights (as the target output depends on reward predicted by the network itself for a new state - see Methods).

Based on the superior learning performance of prospective configuration, we may expect that this learning mechanism has been favored by evolution, thus in the next sections we investigate if it can account for neural activity and behaviour during learning better than backpropagation.

Evidence for prospective configuration: inferring of latent state during learning

Prospective configuration is related to theories proposing that before learning, the brain first infers a latent state of environment from feedback[75–77]. Here, we propose that this inference can be achieved in neural circuits through prospective configuration, where following feedback, neurons in "hidden layers" converge to a prospective pattern of activity that encodes this latent state. We demonstrate that data from various previous studies, which involved the inference of a latent state, can be explained by prospective configuration. These data were previously explained by complex and abstract mechanisms, such as Bayesian models[75,76], while here we mechanistically show with prospective configuration how such inference can be performed by minimal networks encoding only the essential elements of the tasks.

The dynamical inference of latent state from feedback has been recently proposed to take place during sensorimotor learning[76]. In this experiment, participants received different motor perturbations in different contexts, and learned to compensate for these perturbations. Behavioural data suggest that after receiving the feedback, the participants were first employing it to infer the context, and then adapted the force for the inferred context. We demonstrate that prospective configuration is able to reproduce these behavioural data, while backpropagation cannot.

Specifically, in the task (Fig. 5a), participants were asked to move a stick from a starting point to a target point, while experiencing perturbations. The participants experienced a sequence of blocks of trials (Fig. 5c-e) including training, washout, and testing. During the training session, different directions of perturbations, positive (+) or negative (-), were applied in different contexts, blue (B) or red (R) backgrounds, respectively. We denote these trials as B+ and R-. During the washout session, no perturbation was provided. In the testing session, the participants experienced one of the four possible test trials: B+, R+, B-, and R-. To evaluate learning on the test trials, motor adaptation (i.e., the difference between the final and target stick positions) was measured before and after the test trial, on two trials with blue background. The change of the adaptation between these two trials is a reflection of learning about blue context that occurred at the test trial. If participants just associated feedback with the colour of background (B), then the change of adaptation would only occur with test trials B+ and B-. However, experimental data (Fig. 5f, right) show that there was substantial adaptation change also with R+ trials (which was even bigger than with B- trials).

To model learning in this task, we consider a neural network (Fig. 5b) where input nodes encode the colour of background, and outputs encode movement compensations in the two directions. Importantly, this network also includes hidden neurons encoding belief of being in the contexts associated with the two backgrounds ([B] and [R]). Trained with the exact procedure of the experiment[76] from randomly initialized weights, prospective configuration with this minimal network can reproduce the behavioural data, while backpropagation cannot (cf., Fig. 5f left and middle).

Prospective configuration can produce change in adaptation with R+ test trial, because after + feedback, it is able to also activate context [B] that was associated with this feedback during training, and then learn compensation for this latent state. To shed light on how this inference takes place in the model, the bottom parts of Fig. 5c-d show evolution of the weights of the network over sessions (thickness represents the strength of connections). Fig. 5e bottom, shows the difference between the two learning rules at the

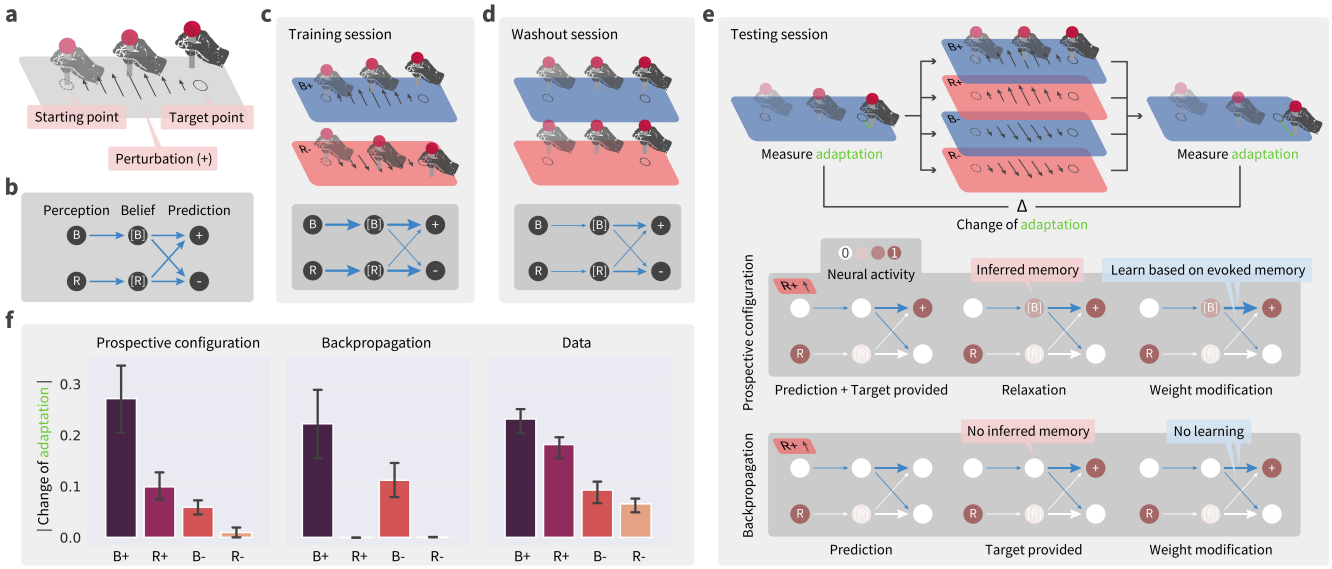


Fig. 5| Prospective configuration explains contextual inference in human sensorimotor learning. ►**a**| The structure of an experimental trial, where participants were asked to move a stick from the starting point to the target point while experiencing perturbations. ►**b**| The minimal network for this task, including six connections encoding the associations from the backgrounds (B and R) to the belief of contexts ([B] and [R]), as well as from belief of contexts to prediction of perturbations (+ and -). ►**c-e**| A sequence of sessions the participants experienced: training, washout, and testing. Inside each panel, the darker box demonstrates the expected network after the session, where thickness represents the strength of connections. In the testing session, the darker box explains how the two learning rules learn differently on the R+ trial, leading to the differences in panel f. ►**f**| Predictions of the two learning rules compared against behavioural data measured from human participants, where prospective configuration reproduces the key patterns of data but backpropagation cannot.

exposure to R+: although B is not perceived, prospective configuration infers a moderate excitation of the belief of blue context [B], because the positive connection from [B] to + was built during the training session. The activation of [B] enables the learning of weights from [B] to + and -; while backpropagation does not modify any weights originating from [B].

Studies of animal conditioning have also observed that feedback in learning tasks involving multiple stimuli may trigger learning about non-presented stimuli[78–82]. For example, in one study[78] rats were trained to associate fear (electric shock) with noise and light; and then, in one group, fear related to light was eliminates in an extinction session (Fig. 6a). Remarkably, the data suggested that eliminating the fear to light increased the fear to noise (Fig. 6b). Such learning is not predicted by the standard Rescorla-Wagner model[83]. We consider a neural network (Fig. 6c) that includes two input neurons encoding the two stimuli, two hidden neurons, and one output neuron encoding the fear. Trained with the exact procedure of animal experiment[78] from randomly initialized weights, prospective configuration with this simple network can reproduce the data, while backpropagation cannot (cf., Fig. 6b blue and orange). In the network employing prospective configuration, the feedback changes the activity of a hidden neuron previously associated with this feedback and with non-presented stimulus (noise), and hence enables modification of connections of this neuron (a learning mechanism analogous to that in sensorimotor learning Fig. 5, see Extended Data Fig. 5 for details).

Evidence for prospective configuration: discovering task structure during learning

Prospective configuration is also able to discover the underlying task structure in reinforcement learning. Particularly, we consider a task where reward probabilities of different options were not independent[75].

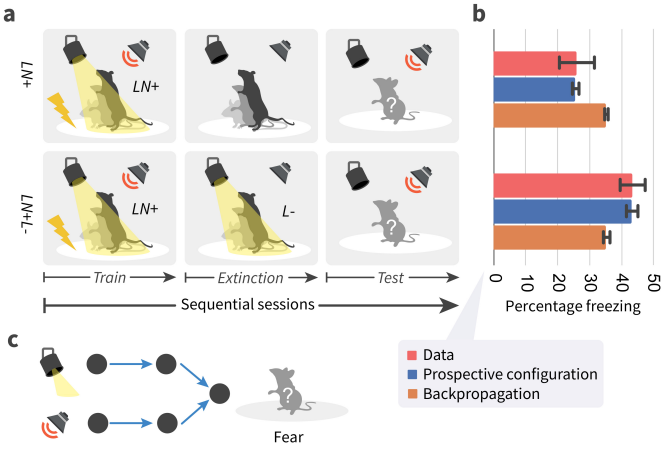


Fig. 6| Prospective configuration infers latent state during fear conditioning. ► a| The fear conditioning task, where rats are first trained to associate fear (electric shock) with noise and light; then in one of the groups, fear related to light is eliminated in extinction session; finally, the predicted fear (percentage of rats freezing) of noise is measures in test session. **► b|** The predicted fear from networks trained with prospective configuration and backpropagation, compared against the fear (percentage freezing) measured in rats. Prospective configuration reproduces the key finding that eliminating the fear to light changes the fear to noise. **► c|** The architecture of simulated networks.

In this study humans were choosing between two options, whose reward probabilities were constrained such that one option had higher reward probability than the other (Fig. 7a). Occasionally the reward probabilities were swapped, so if one probability was increased, the other was decreased by the same amount. Remarkably, the recorded fMRI data suggested that participants learned that the values of two options were negatively correlated, and on each trial updated the value estimates of both options in opposite ways. This conclusion was drawn from the analysis of the signal from medial prefrontal cortex which encoded the expected value of reward. Fig. 7c, right compares this signal after making a choice on two consecutive trials: a trial on which reward was not received ("Punish trial") and the next trial. If the participant selected the same option on both trials ("Stay"), the signal decreased, indicating the reward expected by the participant was reduced. Remarkably, if the participant selected the other option on the next trial ("Switch"), the signal increased, suggesting that negative feedback for one option increased the value estimate for the other. Such learning is not predicted by standard reinforcement learning models[75].

This task can be conceptualized as having a latent state encoding which option is superior, and this latent state determines the reward probabilities for both options. Consequently, we consider a neural network reflecting this structure (Fig. 7b) that includes an input neuron encoding being in this task (equal to 1 in simulations), a hidden neuron encoding the latent state, and two output neurons encoding the reward probabilities for the two options. Trained with the exact procedure of the experiment[75] from randomly initialized weights, prospective configuration with this minimal network can reproduce the data, while backpropagation cannot (cf., Fig. 7c left and middle).

To shed light on the difference between the models, we simulate an "idealized" version of the task in Fig. 7d-e: the network shown in the inset starts from $(\{W_0 = 1, W_1 = 1, W_2 = -1\})$ and is trained for 64 trials in total. The rewards and punishments are delivered deterministically, and the reversal only occurs once at the beginning of training. Fig. 7d inspects prospective configuration at the first few training iterations: during relaxation, the hidden neuron is able to infer its prospective configuration, i.e., negative hidden activity encoding that the rewarded choice has reversed. In Fig. 7e, such inference by prospective configuration results in an increase of W_1: since it has inferred from the punishment that the rewarded choice has reversed to a non-rewarded one, such punishment strengthens the connection from the latent

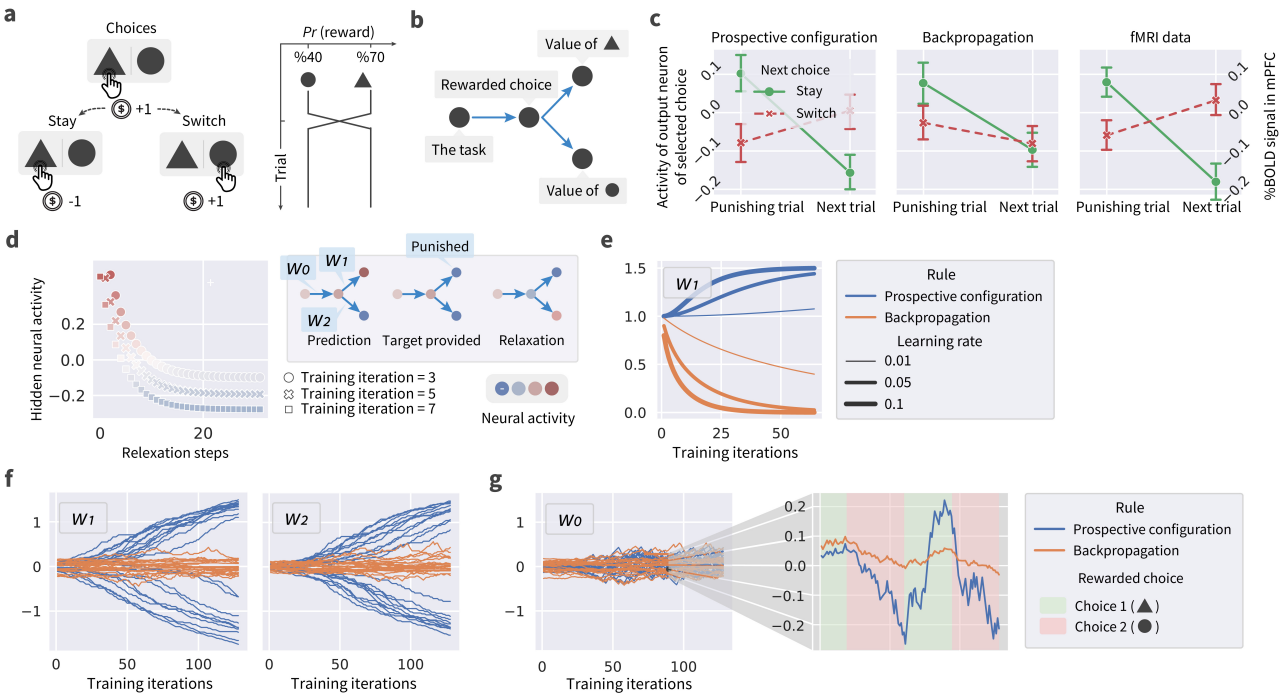


Fig. 7| Prospective configuration can discover the underlying task structure during reinforcement learning. ►**a**| The reinforcement learning task, where human participants need to choose between two options, leading to either reward (gaining coins) or punishment (losing coins) with different probabilities. The probability of reward is occasionally reversed between the two options. ►**b**| The minimal network encoding the essential elements of the task. ►**c**| The activity of output neuron corresponding to the selected option, from networks trained with prospective configuration and backpropagation, compared against the fMRI data measured in human participants, i.e., peak blood oxygenation level-dependent (%BOLD) signal in the medial prefrontal cortex (mPFC). Prospective configuration reproduces the key finding that the expected value (encoded in %BOLD signal in mPFC) increases if the next choice after a punishing trial is to switch to the other option. ►**d**| Prospective configuration at the first few training iterations in an "idealized" version of the task: during relaxation, the hidden neuron is able to infer its prospective configuration, i.e., negative hidden activity encoding that the rewarded choice has reversed. ►**e**| Such inference by prospective configuration results in an increase of W_1. By contrast, in backpropagation W_1 is decreased. Similar behavior also applies to W_2. ►**f**| The W_1 and W_2 in the simulation of the full task with stochastic rewards. Different lines correspond to different simulations. ►**g**| The evolution of W_0 in the full task. In prospective configuration, this weight remains closer to 0 than W_1 and W_2. Inset shows W_0 on one of the simulation in the main plot, where it is demonstrated that prospective configuration easily flips W_0 as the rewarded choice changes, while backpropagation has difficulty in accomplishing this.

state representing non-rewarded choice to a punishment. By contrast, in backpropagation W_1 is decreased: since it receives a punishment without updating the latent state (still encoding that the rewarded choice has not changed), it weakens the connection from the latent state to a reward. Fig. 7f shows the W_1 and W_2 in the simulation of the full task with stochastic rewards. The weights follow a similar pattern as in the simplified task, i.e., their magnitude increases in prospective configuration. This signifies that the network learns that the rewards from the two options are jointly determined by a hidden state. This increase of the magnitude of W_1 and W_2 enables the network to infer the hidden state from the feedback, and learn the task structure (as described for panel b). Fig. 7g shows the evolution of W_0 in the full task. In prospective configuration, this weight remains closer to 0 than W_1 and W_2. Inset shows W_0 on one of the simulation in the main plot, where it is demonstrated that prospective configuration easily flips W_0 as the rewarded choice changes, while backpropagation has difficulty in accomplishing this. The reason of such behavior

is as follows: thanks to large magnitude of W_1 and W_2 in prospective configuration, an error on the output unit results in a large error on the hidden unit, so the network is able to quickly flip the sign of W_0 whenever the observation mismatches the expectation. This results in an increased expectation on the Switch trials (panel c).

Taken together, presented three simulations illustrate that prospective configuration is a common principle that can explain a range of surprising learning effects in diverse tasks.

Discussion

Our paper identifies the principle of prospective configuration, according to which learning relies on neurons first optimizing their pattern of activity to match the correct output, and then reinforcing these prospective activities through synaptic plasticity. Although it was known that in energy-based networks the activity of neurons shifts before weight update, it has been previously thought that this shift is a necessary cost of error propagation in biological networks, and several methods have been proposed to suppress it[29,30,37,50,51] to approximate backpropagation more closely. By contrast, we demonstrate that this reconfiguration of neural activity is the key to achieving learning performance superior to backpropagation, and to explaining experimental data from diverse learning tasks. Prospective configuration further offers a range of experimental predictions distinct from those of backpropagation (Extended Data Figs. 6–7). In sum, we have demonstrated that our novel credit assignment principle of prospective configuration enables more efficient learning than backpropagation by reducing interference, superior performance in situations faced by biological organisms, requires only local computation and plasticity, and can match experimental data across a wide range of tasks.

Our theory addresses a long-standing question of how the brain solves the plasticity-stability dilemma, e.g. how it is possible that despite learning and adjustment of representation in primary visual cortex[84], we can still perceive the world and understand the meaning of visual stimuli we learned over our lifetime. According to prospective configuration, when some weights are modified during learning, compensatory changes are made to other weights, to ensure the stability of previously acquired knowledge. Previous computational models have also proposed mechanisms reducing interference between different pieces of learned information[73,85], and it is highly likely that these mechanisms operate in the brain in addition to prospective configuration and jointly reduce the interference most effectively.

The advantages of prospective configuration suggest that it may be profitably applied in machine learning to improve the efficiency and performance of deep neural networks. An obstacle for this is that the relaxation phase is computationally expensive. However, it has been demonstrated that the speed of energy-based networks can be greatly increased by implementing the relaxation on analog hardware[86,87], potentially resulting in energy-based network being faster than backpropagation. Therefore, we anticipate that our discoveries may change the blueprint of next-generation machine learning hardware — switching from the current digital tensor base to analog hardware, being closer to the brain and potentially far more efficient.

References

1. Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J. & Hinton, G. Backpropagation and the brain. *Nat. Rev. Neurosci.* **21**, 335–346 (2020).

2. Richards, B. A. *et al.* A deep learning framework for neuroscience. *Nat. Neurosci.* **22**, 1761–1770 (2019).

3. Whittington, J. C. & Bogacz, R. Theories of error back-propagation in the brain. *Trends Cogn. Sci.* (2019).

4. Zipser, D. & Andersen, R. A. A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature* **331**, 679–684 (1988).

5. Singer, Y. *et al.* Sensory cortex is optimized for prediction of future input. *Elife* **7**, e31557 (2018).

6. Cadieu, C. F. *et al.* Deep neural networks rival the representation of primate it cortex for core visual object recognition. *PLoS Comput. Biol.* **10**, e1003963 (2014).

7. Yamins, D. L. *et al.* Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proc. Natl. Acad. Sci.* **111**, 8619–8624 (2014).

8. Khaligh-Razavi, S.-M. & Kriegeskorte, N. Deep supervised, but not unsupervised, models may explain it cortical representation. *PLoS Comput. Biol.* **10** (2014).

9. Yamins, D. L. & DiCarlo, J. J. Using goal-driven deep learning models to understand sensory cortex. *Nat. Neurosci.* **19**, 356 (2016).

10. Keller, G. B. & Mrsic-Flogel, T. D. Predictive processing: A canonical cortical computation. *Neuron* **100**, 424–435 (2018).

11. Whittington, J. C. *et al.* The tolman-eichenbaum machine: Unifying space and relational memory through generalization in the hippocampal formation. *Cell* **183**, 1249–1263 (2020).

12. Banino, A. *et al.* Vector-based navigation using grid-like representations in artificial agents. *Nature* **557**, 429–433 (2018).

13. Werbos, P. Beyond regression: new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harv. Univ.* (1974).

14. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning internal representations by error propagation. Tech. Rep., California Univ San Diego La Jolla Inst for Cognitive Science (1985).

15. Parker, D. B. Learning-logic: Casting the cortex of the human brain in silicon. *Tech. report TR-47* (1985).

16. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).

17. Krizhevsky, A., Sutskever, I. & Hinton, G. E. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 25, 1097–1105 (2012).

18. He, H., Boyd-Graber, J., Kwok, K. & Daum III, H. Opponent modeling in deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)* (2016).

19. Hannun, A. *et al.* Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567* (2014).

20. Vaswani, A. *et al.* Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 5998–6008 (2017).

21. Mnih, V. *et al.* Human-level control through deep reinforcement learning. *Nature* (2015).

22. Silver, D. *et al.* Mastering the game of go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016).

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

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

25. Tsividis, P. A., Pouncy, T., Xu, J. L., Tenenbaum, J. B. & Gershman, S. J. Human learning in atari. In *2017 AAAI Spring Symposium Series* (2017).

26. Kell, A. J., Yamins, D. L., Shook, E. N., Norman-Haignere, S. V. & McDermott, J. H. A task-optimized neural network replicates human auditory behavior, predicts brain responses, and reveals a cortical processing hierarchy. *Neuron* **98**, 630–644 (2018).

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

28. Guerguiev, J., Lillicrap, T. P. & Richards, B. A. Towards deep learning with segregated dendrites. *Elife* **6**, e22901 (2017).

29. Scellier, B. & Bengio, Y. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Front. Comput. Neurosci.* **11**, 24 (2017).

30. Whittington, J. C. & Bogacz, R. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural Comput.* **29**, 1229–1262 (2017).

31. Lillicrap, T. P., Cownden, D., Tweed, D. B. & Akerman, C. J. Random synaptic feedback weights support error backpropagation for deep learning. *Nat. Commun.* **7**, 1–10 (2016).

32. Roelfsema, P. R. & Ooyen, A. v. Attention-gated reinforcement learning of internal representations for classification. *Neural Comput.* **17**, 2176–2214 (2005).

33. Pozzi, I., Bohté, S. & Roelfsema, P. A biologically plausible learning rule for deep learning in the brain. *arXiv preprint arXiv:1811.01768* (2018).

34. Körding, K. P. & König, P. Supervised and unsupervised learning with two sites of synaptic integration. *J. Comput. Neurosci.* **11**, 207–215 (2001).

35. Richards, B. A. & Lillicrap, T. P. Dendritic solutions to the credit assignment problem. *Curr. Opin. Neurobiol.* **54**, 28–36 (2019).

36. Payeur, A., Guerguiev, J., Zenke, F., Richards, B. A. & Naud, R. Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. *Nat. Neurosci.* 1–10 (2021).

37. Song, Y., Lukasiewicz, T., Xu, Z. & Bogacz, R. Can the brain do backpropagation?—Exact implementation of backpropagation in predictive coding networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 22566 (Europe PMC Funders, 2020).

38. Salvatori, T., Song, Y., Lukasiewicz, T., Bogacz, R. & Xu, Z. Reverse differentiation via predictive coding. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2022).

39. McCloskey, M. & Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, vol. 24, 109–165 (Elsevier, 1989).

40. McNaughton, B. L. & O'Reilly, R. C. Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of. *Psychol. Rev.* **102**, 419–457 (1995).

41. Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci.* **79**, 2554–2558 (1982).

42. Friston, K. A theory of cortical responses. *Philos. Transactions Royal Soc. B: Biol. sciences* **360**, 815–836 (2005).

43. Rao, R. P. & Ballard, D. H. Predictive coding in the visual cortex: A functional interpretation of some extra-classical receptive-field effects. *Nat. Neurosci.* **2**, 79–87 (1999).

44. Friston, K. The free-energy principle: A unified brain theory? *Nat. Rev. Neurosci.* **11**, 127–138 (2010).

45. Hohwy, J., Roepstorff, A. & Friston, K. Predictive coding explains binocular rivalry: An epistemological review. *Cognition* **108**, 687–701 (2008).

46. Auksztulewicz, R. & Friston, K. Repetition suppression and its contextual determinants in predictive coding. *Cortex* **80**, 125–140 (2016).

47. Watanabe, E., Kitaoka, A., Sakamoto, K., Yasugi, M. & Tanaka, K. Illusory motion reproduced by deep neural networks trained for prediction. *Front. psychology* **9**, 345 (2018).

48. Feldman, H. & Friston, K. Attention, uncertainty, and free-energy. *Front. human neuroscience* **4**, 215 (2010).

49. Kanai, R., Komura, Y., Shipp, S. & Friston, K. Cerebral hierarchies: predictive processing, precision and the pulvinar. *Philos. Transactions Royal Soc. B: Biol. Sci.* **370**, 20140169 (2015).

50. Bengio, Y. & Fischer, A. Early inference in energy-based models approximates back-propagation. *arXiv preprint arXiv:1510.02777* (2015).

51. Millidge, B., Tschantz, A. & Buckley, C. L. Predictive coding approximates backprop along arbitrary computation graphs. *arXiv preprint arXiv:2006.04182* (2020).

52. O'reilly, R. C. & Munakata, Y. *Computational explorations in cognitive neuroscience: Understanding the mind by simulating the brain* (MIT Press Cambridge, 2000).

53. Quilodran, R., Rothe, M. & Procyk, E. Behavioral shifts and action valuation in the anterior cingulate cortex. *Neuron* **57**, 314–325 (2008).

54. Wallis, J. D. & Kennerley, S. W. Heterogeneous reward signals in prefrontal cortex. *Curr. opinion neurobiology* **20**, 191–198 (2010).

55. Buckley, C. L., Kim, C. S., McGregor, S. & Seth, A. K. The free energy principle for action and perception: A mathematical review. *J. Math. Psychol.* **81**, 55–79 (2017).

56. Hinton, G. E. Training products of experts by minimizing contrastive divergence. *Neural computation* **14**, 1771–1800 (2002).

57. LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M. & Huang, F. A tutorial on energy-based learning. *Predict. structured data* **1** (2006).

58. Hinton, G. E. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, 599–619 (Springer, 2012).

59. Felleman, D. J. & Van Essen, D. C. Distributed hierarchical processing in the primate cerebral cortex. *Cereb. Cortex* **1**, 1–47 (1991).

60. Lee, D.-H., Zhang, S., Fischer, A. & Bengio, Y. Difference target propagation. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 498–515 (Springer, 2015).

61. Fontenla-Romero, Ó., Guijarro-Berdiñas, B., Martinez-Rego, D., Pérez-Sánchez, B. & Peteiro-Barral, D. Online machine learning. In *Efficiency and Scalability Methods for Computational Intellect*, 27–54 (IGI Global, 2013).

62. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M. & Bouchachia, A. A survey on concept drift adaptation. *ACM Comput. Surv. (CSUR)* **46**, 1–37 (2014).

63. Hinton, G. E. *et al.* Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, vol. 1, 12 (Amherst, MA, 1986).

64. Hassabis, D., Kumaran, D., Summerfield, C. & Botvinick, M. Neuroscience-inspired artificial intelligence. *Neuron* **95**, 245–258 (2017).

65. Xiao, H., Rasul, K. & Vollgraf, R. Fashion MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

66. Krizhevsky, A. & Hinton, G. Learning multiple layers of features from tiny images. *Report* (2009).

67. Jia, X. *et al.* Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205* (2018).

68. Puri, R., Kirby, R., Yakovenko, N. & Catanzaro, B. Large scale language modeling: Converging on 40gb of text in four hours. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 290–297 (IEEE, 2018).

69. Berner, C. *et al.* Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680* (2019).

70. Ioffe, S. & Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning (ICML)* (2015).

71. Ratcliff, R. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychol. review* **97**, 285 (1990).

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

73. Zenke, F., Poole, B. & Ganguli, S. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, 3987–3995 (PMLR, 2017).

74. Sutton, R. S. & Barto, A. G. *Introduction to Reinforcement Learning*, vol. 2 (MIT Press Cambridge, 1998).

75. Hampton, A. N., Bossaerts, P. & O'doherty, J. P. The role of the ventromedial prefrontal cortex in abstract state-based inference during decision making in humans. *J. Neurosci.* **26**, 8360–8367 (2006).

76. Heald, J. B., Lengyel, M. & Wolpert, D. M. Contextual inference underlies the learning of sensorimotor repertoires. *Nature* **600**, 489–493 (2021).

77. Larsen, T., Leslie, D. S., Collins, E. J. & Bogacz, R. Posterior weighted reinforcement learning with state uncertainty. *Neural computation* **22**, 1149–1179 (2010).

78. Kaufman, M. A. & Bolles, R. C. A nonassociative aspect of overshadowing. *Bull. Psychon. Soc.* **18**, 318–320 (1981).

79. Matzel, L. D., Schachtman, T. R. & Miller, R. R. Recovery of an overshadowed association achieved by extinction of the overshadowing stimulus. *Learn. Motiv.* **16**, 398–412 (1985).

80. Matzel, L. D., Shuster, K. & Miller, R. R. Covariation in conditioned response strength between stimuli trained in compound. *Animal Learn. & Behav.* **15**, 439–447 (1987).

81. Hallam, S. C., Matzel, L. D., Sloat, J. S. & Miller, R. R. Excitation and inhibition as a function of posttraining extinction of the excitatory cue used in pavlovian inhibition training. *Learn. Motiv.* **21**, 59–84 (1990).

82. Miller, R. R., Esposito, J. J. & Grahame, N. J. Overshadowing-like effects between potential comparator stimuli: Covariation in comparator roles of context and punctate excitor used in inhibitory training as a function of excitor salience. *Learn. Motiv.* **23**, 1–26 (1992).

83. Rescorla, R. A. A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. *Curr. research theory* 64–99 (1972).

84. Poort, J. *et al.* Learning enhances sensory and multiple non-sensory representations in primary visual cortex. *Neuron* **86**, 1478–1490 (2015).

85. McClelland, J. L., McNaughton, B. L. & O'Reilly, R. C. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychol. review* **102**, 419 (1995).

86. Foroushani, A. N., Assaf, H., Noshahr, F. H., Savaria, Y. & Sawan, M. Analog circuits to accelerate the relaxation process in the equilibrium propagation algorithm. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1–5 (IEEE, 2020).

87. Hertz, J., Krogh, A., Lautrup, B. & Lehmann, T. Nonlinear backpropagation: Doing backpropagation without derivatives of the activation function. *IEEE Transactions on Neural Networks* **8**, 1321–1327 (1997).

88. Goodfellow, I., Bengio, Y. & Courville, A. *Deep learning* (MIT Press Cambridge, 2016).

89. O'Reilly, R. C. Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural Comput.* **8**, 895–938 (1996).

90. Almeida, L. B. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Artificial Neural Networks: Concept Learning*, 102–111 (IEEE Computer Society Press, 1990).

91. Pineda, F. Generalization of back propagation to recurrent and higher order neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 602–611 (1987).

92. Pineda, F. J. Dynamics and architecture for neural computation. *J. Complex.* **4**, 216–245 (1988).

93. Hebb, D. O. *The organisation of behaviour: A neuropsychological theory* (Science Editions New York, 1949).

94. Glorot, X. & Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256 (JMLR Workshop and Conference Proceedings, 2010).

95. Krizhevsky, A., Nair, V. & Hinton, G. CIFAR-10. In *Canadian Institute for Advanced Research*, vol. 5, 4 (2010).

96. Žliobaitė, I. Learning under concept drift: An overview. *arXiv preprint arXiv:1010.4784* (2010).

97. Tsymbal, A. The problem of concept drift: Definitions and related work. *Comput. Sci. Dep. Trinity Coll. Dublin* **106**, 58 (2004).

98. Sutton, R. S. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 8 (1995).

99. Geramifard, A., Dann, C., Klein, R. H., Dabney, W. & How, J. P. RLPy: A value-function-based reinforcement learning framework for education and research. *J. Mach. Learn. Res.* **16**, 1573–1578 (2015).

100. Moore, A. Efficient memory-based learning for robot control. Tech. Rep., Carnegie Mellon University, Pittsburgh, PA (1990).

101. Barto, A. G., Sutton, R. S. & Anderson, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Syst. Man, Cybern.* 834–846 (1983).

102. Brockman, G. *et al.* Openai gym. *arXiv:1606.01540* (2016).

103. Welford, B. Note on a method for calculating corrected sums of squares and products. *Technometrics* **4**, 419–420 (1962).

104. Knuth, D. E. *Art of computer programming, volume 2: Seminumerical algorithms* (Addison-Wesley Professional, 2014).

105. Meulemans, A., Carzaniga, F. S., Suykens, J. A., Sacramento, J. & Grewe, B. F. A theoretical framework for target propagation. *arXiv preprint arXiv:2006.14331* (2020).

106. Bengio, Y. Deriving differential target propagation from iterating approximate inverses. *arXiv preprint arXiv:2007.15139* (2020).

107. Bastos, A. M. *et al.* Canonical microcircuits for predictive coding. *Neuron* **76**, 695–711 (2012).

108. Attinger, A., Wang, B. & Keller, G. B. Visuomotor coupling shapes the functional development of mouse visual cortex. *Cell* **169**, 1291–1302 (2017).

109. Boerlin, M., Machens, C. K. & Denève, S. Predictive coding of dynamical variables in balanced spiking networks. *PLoS Comput. Biol.* **9**, e1003258 (2013).

110. Brendel, W., Bourdoukan, R., Vertechi, P., Machens, C. K. & Denéve, S. Learning to represent signals spike by spike. *PLoS Comput. Biol.* **16**, e1007692 (2020).

111. Heekeren, H. R., Marrett, S., Bandettini, P. A. & Ungerleider, L. G. A general mechanism for perceptual decision-making in the human brain. *Nature* **431**, 859–862 (2004).

112. Krotov, D. & Hopfield, J. J. Dense associative memory for pattern recognition. In *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 29, 1172–1180 (2016).

113. Soto, V., Suárez, A. & Martínez-Muñoz, G. An urn model for majority voting in classification ensembles. In *Advances in Neural Information Processing Systems (NeurIPS)* (Neural Information Processing Systems Foundation, 2016).

114. Bengio, Y., Mesnard, T., Fischer, A., Zhang, S. & Wu, Y. Stdp as presynaptic activity times rate of change of postsynaptic activity. *arXiv preprint arXiv:1509.05936* (2015).

115. Penrose, R. A generalized inverse for matrices. In *Mathematical proceedings of the Cambridge philosophical society*, vol. 51, 406–413 (Cambridge University Press, 1955).

Methods

This section provides necessary details for replication of results in the main text.

Models

Throughout this work, we compare the established theory of *backpropagation* to the proposed new principle of *prospective configuration*. As explained in the main text, backpropagation is used to train *artificial neural networks* (ANNs), where the activity of a neuron is *fixed* to a value based on its input, while prospective configuration occurs in *energy-based networks* (EBNs), where the activity of a neuron is *not* fixed.

Since in ANNs the activity of neurons $\boldsymbol{x}$ is determined by their input, the output of the network can be obtained by propagating the inputs "forward" through the computational graph. The output can then be compared against a target pattern to get a measure of difference known as a *loss*. Since the value of a node (activity of a neuron) in the computational graph is explicitly computed as a function of its input, the computational graph is usually differentiable. Thus, training ANNs with backpropagation modifies the weights $\boldsymbol{w}$ to take a step towards the negative gradient of loss L,

$$\Delta \boldsymbol{w} \sim -\frac{\partial L}{\partial \boldsymbol{w}}, \tag{1}$$

during which the activity of neurons $\boldsymbol{x}$ is fixed. The weights $\boldsymbol{w}$ requiring modification might be many steps away from the output on the computational graph, where the loss L is computed; thus, $\frac{\partial L}{\partial \boldsymbol{w}}$ is often obtained by applying the chain rule of computing a derivative through intermediate variables (activity of output and hidden neurons). For example, consider a network with 4 layers and let $\boldsymbol{x}^l$ denote the activity of neurons in layer l, while $\boldsymbol{w}^l$ denote the weights of connections between layers l and $l+1$. Then the change in the weights originating from the first layer is computed: $\frac{\partial L}{\partial \boldsymbol{w}^1} = \frac{\partial L}{\partial \boldsymbol{x}^4} \cdot \frac{\partial \boldsymbol{x}^4}{\partial \boldsymbol{x}^3} \dots \frac{\partial \boldsymbol{x}^2}{\partial \boldsymbol{w}^1}$. This enables the loss to be backpropagated through the graph to provide a direction of update for all weights.

In contrast to ANNs, in EBNs, the activity of neurons $\boldsymbol{x}$ is not fixed to the input from a previous layer. Instead, an energy function E is defined as a function of the neural activity $\boldsymbol{x}$ and weights $\boldsymbol{w}$. For networks organized in layers (considered in this paper), the energy can be decomposed into a sum of local energy terms E^l:

$$E = \sum_l E^l \left(\boldsymbol{x}^l, \boldsymbol{w}^{l-1}, \boldsymbol{x}^{l-1} \right). \tag{2}$$

Here, E^l is called local energy, because it is a function of $\boldsymbol{x}^l$, $\boldsymbol{x}^{l-1}$, and $\boldsymbol{w}^{l-1}$ that are neighbours and connected to each other. This ensures that the optimization of energy E can be implemented by local circuits, because the derivative of E with respect to any neural activity (or weights) results in an equation containing only the local activity (or weights) and the activity of adjacent neurons. Predictions with EBNs are computed by clamping the input neurons to an input pattern, and then modifying the activity of all other neurons to decrease the energy:

$$\Delta \boldsymbol{x} \sim -\frac{\partial E}{\partial \boldsymbol{x}}. \tag{3}$$

Since the terms in E can be divided into local energy terms, this results in an equation that can be implemented with local circuits. This process of modifying the neural activity to decrease the energy is called *relaxation*, and we refer to the equation describing relaxation as *neural dynamics* — because it describes the dynamics of the neural activity in EBNs. After convergence of relaxation, the activities of

the output neurons are taken as the prediction made by the EBN. Different EBNs are trained in slightly different ways. In case of *predictive coding network*[30,43,55] (PCN), training involves clamping the input and output neurons to input and target patterns, respectively. Then, relaxation is run until convergence ($\boldsymbol{x} = \overset{*}{\boldsymbol{x}}$), after which the weights are updated using the activity at convergence to further decrease the energy:

$$\Delta \boldsymbol{w} \sim -\frac{\partial E}{\partial \boldsymbol{w}}\bigg|_{\boldsymbol{x}=\overset{*}{\boldsymbol{x}}}. \tag{4}$$

This will also result in an equation that can be implemented with local plasticity since it is just a gradient descent on the local energy. We refer to such an equation as *weight dynamics*, because it describes the dynamics of the synaptic weights in EBNs.

Backpropagation and prospective configuration are not restricted to specific models. Depending on the structure of the network, and the choice of the energy function, one can define different models that implement the principle of backpropagation or prospective configuration. In the main text and most of the Extended Data, we investigate the most standard layered network. In this case, both ANNs and EBNs include L layers of weights $\boldsymbol{w}^1, \boldsymbol{w}^2, \ldots, \boldsymbol{w}^L$, and $L+1$ layers of neurons $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^{L+1}$, where $\boldsymbol{x}^1$ and $\boldsymbol{x}^{L+1}$ are the input and output neurons, respectively. We consider the relationship between activities in adjacent layers for ANNs given by

$$\boldsymbol{x}^l = \boldsymbol{w}^{l-1} f\left(\boldsymbol{x}^{l-1}\right), \tag{5}$$

and the energy function for EBNs described by

$$E^l = \frac{1}{2}\left(\boldsymbol{x}^l - \boldsymbol{w}^{l-1} f\left(\boldsymbol{x}^{l-1}\right)\right)^2. \tag{6}$$

This defines the ANNs to be the standard *multilayer perceptrons* (MLPs) and the EBNs to be the PCN. In Eq. (6) and below, $(\boldsymbol{v})^2$ denotes the inner product of vector $\boldsymbol{v}$ with itself. The comparison between backpropagation and prospective configuration in the main text is thus between the above MLPs and PCNs. This choice is justified by that (1) they are the most standard models[88] and also (2) it is established that they two are closely related[30,37] (i.e., they make the same prediction with the same weights and input pattern), thus enabling a fair comparison. Nevertheless, we show that the theory (Extended Data Fig. 1) and empirical comparison (Extended Data Figs. 3 and 4) between backpropagation and prospective configuration generalize to other choices of network structures and energy functions, i.e., other EBNs and ANNs, such as *GeneRec*[89] and *Almeida-Pineda*[90–92].

Putting Eqs. (5) and (6) into the general framework, we can obtain the equations that describe MLPs and PCNs, respectively. Assume the input and target patterns are $\boldsymbol{s}^{\text{in}}$ and $\boldsymbol{s}^{\text{target}}$, respectively. Prediction with MLPs is:

$$\boldsymbol{x}^1 = \boldsymbol{s}^{\text{in}} \text{ and } \boldsymbol{x}^l = \boldsymbol{w}^{l-1} f\left(\boldsymbol{x}^{l-1}\right) \text{ for } l > 1, \tag{7}$$

where $\boldsymbol{x}^{L+1}$ is the prediction. Training MLPs with backpropagation is described by:

$$\Delta \boldsymbol{w}^l \sim -\frac{\partial L}{\partial \boldsymbol{w}^l} = -\frac{\partial L}{\partial \boldsymbol{x}^{L+1}} \cdot \frac{\partial \boldsymbol{x}^{L+1}}{\partial \boldsymbol{x}^L} \cdots \frac{\partial \boldsymbol{x}^{l+1}}{\partial \boldsymbol{w}^l} \text{ where } L = \frac{1}{2}\left(\boldsymbol{x}^{L+1} - \boldsymbol{s}^{\text{target}}\right)^2, \tag{8}$$

which backpropagates the error $\frac{\partial L}{\partial \boldsymbol{x}^l}$ layer by layer from output neurons.

The neural dynamics of PCNs can be obtained using Eq. (2):

$$\Delta \boldsymbol{x}^l \sim -\frac{\partial E}{\partial \boldsymbol{x}^l} = -\frac{\partial (E^l + E^{l+1})}{\partial \boldsymbol{x}^l}. \tag{9}$$

Similarly, the weight dynamics of PCNs can be found:

$$\Delta \boldsymbol{w}^l \sim -\frac{\partial E}{\partial \boldsymbol{w}^l} = -\frac{\partial E^{l+1}}{\partial \boldsymbol{w}^l}. \tag{10}$$

To reveal the neural implementation of PCN, we define the prediction errors to be

$$\boldsymbol{\varepsilon}^l = \boldsymbol{x}^l - \boldsymbol{w}^{l-1} f\left(\boldsymbol{x}^{l-1}\right). \tag{11}$$

The neural and weight dynamics of PCN can be expressed (by evaluating derivatives in Eqs. (9) and (10)):

$$\Delta \boldsymbol{x}^l \sim -\boldsymbol{\varepsilon}^l + f'\left(\boldsymbol{x}^l\right) \circ \left(\boldsymbol{w}^l\right)^T \boldsymbol{\varepsilon}^{l+1} \tag{12}$$

$$\Delta \boldsymbol{w}^l \sim \boldsymbol{\varepsilon}^{l+1} \left(f\left(\boldsymbol{x}^l\right)\right)^T, \tag{13}$$

where the symbol $\circ$ denotes element-wise multiplication. Assuming that $\boldsymbol{\varepsilon}^l$ and $\boldsymbol{x}^l$ are encoded in the activity of error and value neurons, respectively, Eqs. (11) and (12) can be realized with the neural implementation in Fig. 2c bottom. Particularly, error $\boldsymbol{\varepsilon}$ and value $\boldsymbol{x}$ neurons are represented by red and blue nodes, respectively; excitatory $+$ and inhibitory $-$ connections are represented by connections with solid and hollow nodes, respectively. Thus, Eqs. (11) and (12) are implemented with red and blue connections, respectively. It should also be noticed that the weight dynamics is also realized locally: weight change described by Eq. (13) corresponds to simple Hebbian plasticity[93] in the neural implementation of Fig. 2c bottom, i.e., the change in a weight is proportional to the product of activity of pre-synaptic and post-synaptic neurons. Thus, a PCN, as an EBN, can be implemented with local circuits only, due to the local nature of energy terms (as argued earlier in this section).

In all simulations in this paper (unless stated otherwise), the integration step of the neural dynamics in Eq. (12) (i.e., relaxation) is 0.1, and the relaxation is performed for 128 steps. During the relaxation, if the overall energy is not decreased from the last step, the integration step is reduced by 50%; if the integration step is reduced two times (i.e., reaching 0.025), the relaxation is terminated early. By monitoring the number of relaxation steps performed, we notice that in most of the tasks we performed, the relaxation is terminated early at around 60 iterations.

In the Extended Data, we also investigate other choices of network structures and energy functions, resulting in other ANNs and EBNs. Overall, the EBNs investigated include PCNs[30,43,55], target-PCNs, and *GeneRec*[89], and the ANNs investigated include backpropagation and *Almeida-Pineda*[90–92]. Details of all the models can be found in corresponding previous work, and are also given in the Supplementary Materials (SI) 2.1.

Interference and measuring interference (i.e., target alignment) (Fig. 3)

In Fig. 3a, since it simulates the example in Fig. 1, structure of the network is 1-1-2; weights are all initialized to 1; input pattern is $[1]$ and target pattern is $[0, 1]$. Learning rates of both learning rules are 0.2, and the weights are updated for 24 iterations.

In Fig. 3d, there are 64 neurons in each layer (including input and output layers) for each network; weights are initialized via Xavier uniform initialization[94]. No activation function is used, i.e., linear networks are investigated. Depths of networks (L) are searched from $\{1,2,4,6,8,10,12,14,15\}$, as reported on the x-axis. Input and target patterns are a pair of randomly generated patterns of mean 0 and standard deviation 1. Learning rates of both learning rules are 0.001. Weights are updated for one iteration and target alignment is measured for this iteration for each of the 64 datapoints, then averaged over the 64 datapoints to produce the reported target alignment value. The whole experiment is repeated 3 times and the error bars report the standard error.

Simulations in Fig. 3e-f follow the setup of experiments in Fig. 4, thus, are described at the end of the next section.

Biologically relevant tasks (Fig. 4)

In all supervised learning simulations in Fig. 4a-h, models are trained and tested on two datasets, FashionMNIST[65] and CIFAR-10[95]. With FashionMNIST, models are trained to perform classification of gray-scaled fashion item images into 10 categories such as trousers, pullovers and dresses. FashionMNIST is chosen because it is of moderate and appropriate difficulty for multi-layer non-linear deep neural networks, so that the comparisons with EBNs are informative. Classification of data in CIFAR-10 is more difficult, as it contains colored natural images belonging to categories such as cars, birds and cats. Both datasets consist of 60000 training examples (i.e., training set) and 10000 test examples (i.e., test set).

The experiments in Fig. 4a-h follow the configurations below, except for the parameters investigated in specific panels (such as batch size, size of the dataset, and size of the architecture), which are adjusted as stated in the description of specific experiments. The neural network is composed of 4 layers and 32 hidden neurons in each hidden layer. The size of the input layer is 28×28 for FashionMNIST[65] and 32×32 for CIFAR-10[66] (both datasets are gray-scaled). The size of the output layer is 10, as the number of classes for both datasets. The weights are initialized from a normal distribution with mean 0 and standard deviation $\sqrt{\frac{2}{n^l+n^{l+1}}}$, where n^l and n^{l+1} are the number of neurons of the layer before and after the weight, respectively. This initialization is known as Xavier normal initialization[94]. The activation function $f()$ is *Sigmoid*. We define one *iteration* as updating the weights for one step based on a mini-batch. The number of examples in a mini-batch, called the batch-size, is by default 32. One *epoch* comprises presenting the entire training set, split over multiple mini-batches. At the end of each epoch, the model is tested on the test set and the classification error is recorded as the "test error" of this epoch. The neural network is trained for 64 epochs; thus, ending up with 64 test errors. The mean of the test error over epochs, i.e., during training progress, is an indicator of how fast the model learns. The minimum of the test errors over epochs is an indicator of how well the model can learn, ignoring the possibility of over-fitting due to training for too long. Learning rates are searched independently for each configuration and each model. Each experiment is repeated 10 times (unless stated otherwise), and the error bars represent standard error.

We now describe settings specific to individual experiments. In Fig. 4b different batch sizes are tested (as shown on x-axis). In Fig. 4c the batch size is set to 1. In Fig. 4d different numbers of data points per class (data points per class) are included into the training set (subsets are randomly selected according to different seeds). In Figs. 4e–f investigating concept drifting[62,96,97], changes to class labels are made every 512 epochs, and the models are trained for 4096 epochs in total. Thus, every 512 epochs, 5 out of 10 output neurons are selected, and the mapping from these 5 output neurons to the semantic meaning is pseudo-randomly shuffled. In continual learning of Fig. 4g, training alternates between two tasks. Task 1 is classifying five randomly selected classes in a dataset, and task 2 is classifying the remaining five classes. The whole network is shared by the two tasks, thus, differently from the network used in other

panels, the network only has 5 output neurons. This better corresponds to continual learning with multiple tasks in nature, because, for example, if humans learn to perform two different tasks, they typically use the one brain and one pair of hands (i.e., the whole network is shared), since they do not have two different pairs of hands (i.e., humans share the output layers across tasks). Task 1 is trained for 4 iterations and then task 2 is trained for 4 iterations, and the training continues until total of 84 iterations is reached. After each iteration, error on the test set of each task is measured, as "test error". In Fig. 4h, the mean of test error of both tasks during training of Fig. 4g at different learning rates is reported.

In Fig. 3e, networks of 15 layers are trained and tested on FashionMNIST[65] dataset. Fig. 3f investigates other network depths ($\{1,2,4,6,8,10,12,14,15\}$) in the same setup. Hidden layers are always of size 64. In this experiment, only part of the training set was used (60 datapoints per class) so that the test error is evaluated more frequently to reflect the difference on efficiency of the investigated learning rules. The activation function $f()$ used is *LeakyReLU*, instead of the standard Sigmoid, because Sigmoid results in difficulty in training deep neural networks. Other unmentioned details follows the defaults as described above.

In the reinforcement learning experiments (Fig. 4i), we evaluate performance on three classic reinforcement learning problems: Acrobot[98,99], MountainCar[100], and CartPole[101]. We interact with these environments via a unified interface by OpenAI Gym[102]. The observations s_t of these environments are vectors describing the status of the system, such as velocities and positions of different moving parts (for details refer to the original articles or documentation from OpenAI Gym). Each entry of the observation s_t is normalized to mean 0 and standard deviation 1 via Welford's online algorithm[103,104]. The action space of these environments is discrete. Thus, we can have a network taking in observation s_t and predicting the value (Q) of each action a_t with different output neurons. Such a network is known as an action-value network, in short, a Q network. In our experiment, the Q network contains two hidden layers, each of which contains 64 neurons, initialized the same way as the network used for supervised learning, described before. One can acquire the value of an action a_t at a given observation s_t by feeding in s_t to the Q network and reading out the prediction on the output neuron corresponds to the action a_t, such value is denoted by $Q(s_t, a_t)$. The training of Q is a simple regression problem to target $\hat{R}_t$, obtained via Q-learning with experience replay (summarized in Algorithm 1). Considering s_t to be $\boldsymbol{s}^{\text{in}}$ and $\hat{R}_t$ to be $\boldsymbol{s}^{\text{target}}$, the Q network can be trained with prospective configuration or backpropagation. Note that $\hat{R}_t$ is the target of the selected action a_t (i.e., the target of one of the output neurons corresponds to the selected action a_t), thus, $\hat{R}_t$ is in practice considered to be $\boldsymbol{s}^{\text{target}}[a_t]$. For prospective configuration, it means the rest of the output neurons except the one corresponding to a_t are freed; for backpropagation, it means the error on these neurons are masked out.

PCN of slightly different settings from the defaults is used for prospective configuration: the integration step is fixed to be half of the default (=0.05), and relaxation is performed for a fixed and smaller number of steps (=32). This change is introduced because Q-learning is more unstable (so smaller integration step) and more expensive (so smaller number of relaxation steps) than supervised learning tasks. To produce a smoother curve of "Sum of rewards per episode" in Fig. 4i from the *SumRewardPerEpisode* in Algorithm 1, the *SumRewardPerEpisode* curve along *TrainingEpisode* are averaged with a sliding window of length 200. Each experiment is repeated with 3 random seeds and the shadows represents standard error across them. Learning rates are searched independently for each environment and each model from the range $\{0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001\}$; and the results reported in Fig. 4i are for the learning rates yielding the highest mean of "Sum of rewards per episode" over training episodes.

Algorithm 1: Q-learning with experience replay.

Input: Action-value network Q.
Result: Trained action-value network Q.

Initialize experience replay $\mathscr{R}$ of capacity 50000;
for $TrainingEpisode = 0$; $TrainingEpisode < 10000$; $TrainingEpisode = TrainingEpisode + 1$ **do**
  $\rho = \max(0.01, 0.08 - 0.01 * (TrainingEpisode/200))$; // Anneal probability of exploring
  Get initial observation s_t and set episode termination signal $d_t = False$;
  Initialize $SumRewardPerEpisode = 0$;
  while ! d_t **do** // Collect experience
    With probability ρ sample a random action a_t, otherwise select $a_t = \arg\max_a Q(s_t, a)$;
    Execute a_t, observe reward r_t, new observation r_{t+1} and d_t;
    Accumulate $SumRewardPerEpisode += r_t$;
    Store transition $(s_t, a_t, r_t, s_{t+1}, d_t)$ in $\mathscr{R}$;
    Set $s_t = s_{t+1}$;
  end
  if $length(\mathscr{R}) > 2000$ **then** // Replay and train
    for $epoch = 0$; $epoch < 10$; $epoch = epoch + 1$ **do**
      Sample random minibatch (size=60) of $(s_t, a_t, r_t, s_{t+1}, d_t)$ from $\mathscr{R}$;
      $\hat{R}_t = \begin{cases} r_t, & \text{if } d_t == True \\ r_t + 0.98 \max_a Q(s_{t+1}, a), & \text{otherwise} \end{cases}$;
      Set $\boldsymbol{s}^{\text{in}} = s_t$;
      Set $\boldsymbol{s}^{\text{target}}[a_t] = \hat{R}_t$;
      Train Q with $\boldsymbol{s}^{\text{in}}$ and $\boldsymbol{s}^{\text{target}}[a_t]$ with prospective configuration or backpropagation;
    end
  end
  Report $SumRewardPerEpisode$;
end

Simulation of motor learning (Fig. 5)

As shown in Fig. 5, we train a network that includes 2 input, 2 hidden, and 2 output neurons. The two input neurons are one-to-one connected to the two hidden neurons, and the two hidden neurons are fully connected to the two output neurons. The two input neurons are considered to encode presenting the blue and red background, respectively. The two output neurons are considered to encode the prediction of the perturbations towards positive and negative directions, respectively. Presenting or not presenting a background color are encoded as 1 and 0 respectively; presenting or not presenting perturbations of a particular direction are encoded as 1 and 0, respectively. The weights are initialized from a normal distribution with mean 0 and standard deviation fitted to behavioural data (see below), simulating that the participants have not built any associations before the experiments. Learning rates are independent for the two layers, as we expect the connections from perception to belief and the connections from belief to predictions to have different degree of plasticity. The two learning rates are also fitted to the data (see below).

The number of participants, training and testing trials follow exactly the human experiment[75]. In particular, for each of 24 simulated participants, the weights are initialized with a different seed of the

random number generator. They each experience two stages: training and testing. Note that the pre-training stage performed in the human experiment is not simulated here as its goal was to make human participants familiar with the setup and devices.

In the training stage, the model experiences 24 blocks of trials. In each block, the model is presented with the following sequence of trials, matching the original experiment[75].

- The model is trained with two trials without perturbation: B0 and R0, with order counterbalanced across consecutive blocks. Note that in the human experiment there were two trial types without perturbations (channel and washout trials), but they are simulated in the same way here as B0 or R0 trials, because they both did not include any perturbations.
- The model is trained with 32 trials with perturbations, where there are equal number of B+ and R- within each 8 trials in a pseudorandom order.
- The model experiences two trials: B0 and R0, with order counterbalanced across consecutive blocks.
- The model experiences $n \leftarrow \{14, 16, 18\}$ washout trials (equal number of B0 and R0 trials in a pseudorandom order), where $n \leftarrow \{a, b, c\}$ denotes sampling without replacement from a set of values a, b and c, and replenishing the set whenever becomes empty.
- The model experiences one triplet, where the exposure trial is either B+ or R-, counterbalanced across consecutive blocks. Here, a triplet consists three sequential trials: B0, the specified exposure trial and again B0.
- The model experiences again $n \leftarrow \{6, 8, 10\}$ washout trials (equal number of B0 and R0 trials in a pseudorandom order).
- The model experiences again one triplet, where the exposure trial is either B+ or R-, whichever was not used on the previous triplet.

Then, in the testing stage, the model experiences 8 repetitions of four blocks of trials. In each block, one of combinations B+, R+, B- and R- is tested. The order of the four blocks is shuffled in each of the 8 repetitions. In each block, the model first experiences $n \leftarrow \{2, 4, 6\}$ washout trials (equal number of B0 and R0 trials in a pseudorandom order). Then the model experiences a triplet of trials, where the exposure trial is the combination (B+, R+, B- or R-) tested in a given block, to assess single trial learning of this combination. The change in adaption in the model is computed as the absolute value of the difference in the predictions of perturbations on the two B0 trials in the above triplet, where the prediction of perturbation is computed as the difference between the activities of the two output neurons. The predictions are averaged over participants and the above repetitions.

The parameters of each learning rule are chosen such that the model best reproduces the change in adaptation shown in Fig 5f. In particular, we minimize the sum over set C of the 4 exposure trial types of the squared difference between average change in adaptation in experiment (d_c) and in the model (x_c):

$$\sum_{c \in C} (a x_c - d_c)^2 \tag{14}$$

The model predictions are additionally scaled by a coefficient a fitted to the data, because the behavioural data and model outputs have different scales. Exhaustive search was performed over model parameters: standard deviation of initial weights and two learning rates for two layers could take values from $\{0.01, 0.05, 0.1\}$. Then, for each learning rule and each combination of the above model parameters, the coefficient a is resolved analytically (restricted to be positive) to minimize the sum of the squared errors of Eq. (14).

Simulation of fear conditioning (Fig. 6)

As shown in Fig. 6c, the simulated network includes 2 input, 2 hidden, and 1 output neurons. The weights are initialized from a normal distribution of mean 0 and standard deviation 0.01, reflecting that the animals

have not built an association between stimulus and electric shock before the experiments. Presenting or not presenting the stimulus (noise, light, or shock) is encoded as 1 and 0, respectively. The two input neurons are considered to be the visual and auditory neurons; thus, their activity corresponds to perceiving light and noise, respectively. The output neuron is considered to encode the prediction of the electric shock. The training and extinction sessions are both simulated for 32 iterations with the learning rate of 0.01. In the test session, the model makes a prediction with the presented stimulus (noise only). As in the previous section, we denote by x_c the prediction for each group c from a set $C = \{N+, LN+, LN+L-\}$. To map the prediction to the percentage of freezing, it is scaled by a coefficient a (as the neural activity and the measure of freezing have different units) and shifted by a bias b (as the rats may have some tendency to freeze after salient stimuli even if they had not been associated with a shock). The numbers reported in Fig. 6b are these scaled predictions. The coefficient a (constrained to be positive) and bias b are optimized for prospective configuration and backpropagation independently, analogously as described in the previous section, i.e. their values that minimize summed squared error given below are found analytically.

$$\sum_{c \in C} (ax_c + b - d_c)^2 \tag{15}$$

Simulation of human reinforcement learning (Fig. 7)

As shown in Fig. 7b, we train a network that includes 1 input, 1 hidden, and 2 output neurons. The input neuron is considered to encode being in the task, so it is set to 1 throughout the simulation. The two output neurons encode the prediction of the value of the two choices. Reward and punishment are encoded as 1 and -1, respectively, because the participants were either winning or losing money. The model selects actions stochastically based on the predicted value of the two choices (encoded in the activity of two output neurons) according to the softmax rule (with temperature of 1). The weights are initialized from a normal distribution of mean 0 and standard deviation fitted to experimental data (see below), simulating that the human participants have not built any associations before the experiments. Number of simulated participants (number of repetitions with different seeds) was set to 16 as in the human experiment[75]. The number of trials is not mentioned in the original paper, so we simulate for 128 trials for both learning rules.

To compare the ability of the two learning rules to account for the pattern of signal from mPFC, for each of the rules, we optimized the parameters describing how the model is set up and learns (the standard deviation of initial weights and the learning rate). Namely, we searched for the values of these parameters for which the model produces the most similar pattern of its output activity to that in the experiment. In particular, we minimized the sum over set C of four trial types in Fig. 7c of the squared difference between model predictions x_c and data d_c on mean mPFC signal (Eq. (15)). The model predictions are additionally scaled by a coefficient a and offset by a bias b, because the fMRI signal had different units and baseline than the model. To compute the model prediction for a given trial type, the activity of the output neuron corresponding to the chosen option is averaged across all trials of this type in the entire simulation. The scaled average activity from the model is plotted in Fig. 7c, where the error bars show the standard error of the scaled activity. To fit the model to experimental data, the values of model parameters and the coefficient were found analogously as described in the previous section. In particular, we employ exhaustive grid search on the parameters. The models is simulated for all possible combinations of standard deviation of initial weights, and the learning rate, from the following set: $\{0.01, 0.05, 0.1\}$. Then, for each learning rule and each combination of the above model parameters, the coefficient a (restricted to be positive) and the bias b are resolved analytically to minimize sum of the squared error of Eq. (15).

Data availability

Learning tasks analysed in Fig. 4a-h were built using the publicly available FashionMNIST[65] and CIFAR-10[66] datasets. They are incorporated in most machine learning libraries, and their original releases are available at https://github.com/zalandoresearch/fashion-mnist and https://www.cs.toronto.edu/~kriz/cifar.html, respectively. Reinforcement learning tasks analysed in Fig. 4i were built using the publicly available simulators by OpenAI Gym[102].

Code availability

Complete code and full documentation reproducing all simulation results will be made publicly available at https://github.com/YuhangSong/A-New-Perspective upon publication of this work. It will be released under GNU General Public License v3.0 without any additional restrictions (for license's details see https://opensource.org/licenses/GPL-3.0 by the open source initiative).

Acknowledgements

We thank Timothy Behrens for comments on the manuscript, and Andrew Saxe for discussions. Yuhang Song was supported by the China Scholarship Council under the State Scholarship Fund and J.P. Morgan AI Research Awards. Beren Millidge and Rafal Bogacz were supported by the the Biotechnology and Biological Sciences Research Council grant BB/S006338/1 and Medical Research Council grant MC_UU_00003/1. Thomas Lukasiewicz and Tommaso Salvatori were supported by the Alan Turing Institute under the EPSRC grant EP/N510129/1 and by the AXA Research Fund. Zhenghua Xu was supported by National Natural Science Foundation of China under the grant 61906063, by the Natural Science Foundation of Hebei Province, China, under the grant F2021202064, by the Natural Science Foundation of Tianjin City, China, under the grant 19JCQNJC00400, by the "100 Talents Plan" of Hebei Province, China, under the grant E2019050017, and by the Yuanguang Scholar Fund of Hebei University of Technology, China.

Author contributions

Y.S. and R.B. conceived the project. Y.S., R.B., B.M. and T.S. contributed ideas for experiments and analysis. Y.S. and B.M. performed simulation experiments and analysis. Y.S., T.L, and R.B. managed the project. T.L, and Z.X. advised on the project. Y.S., R.B. and B.M. wrote the paper. T.S., T.L, and Z.X. provided revisions to the paper.

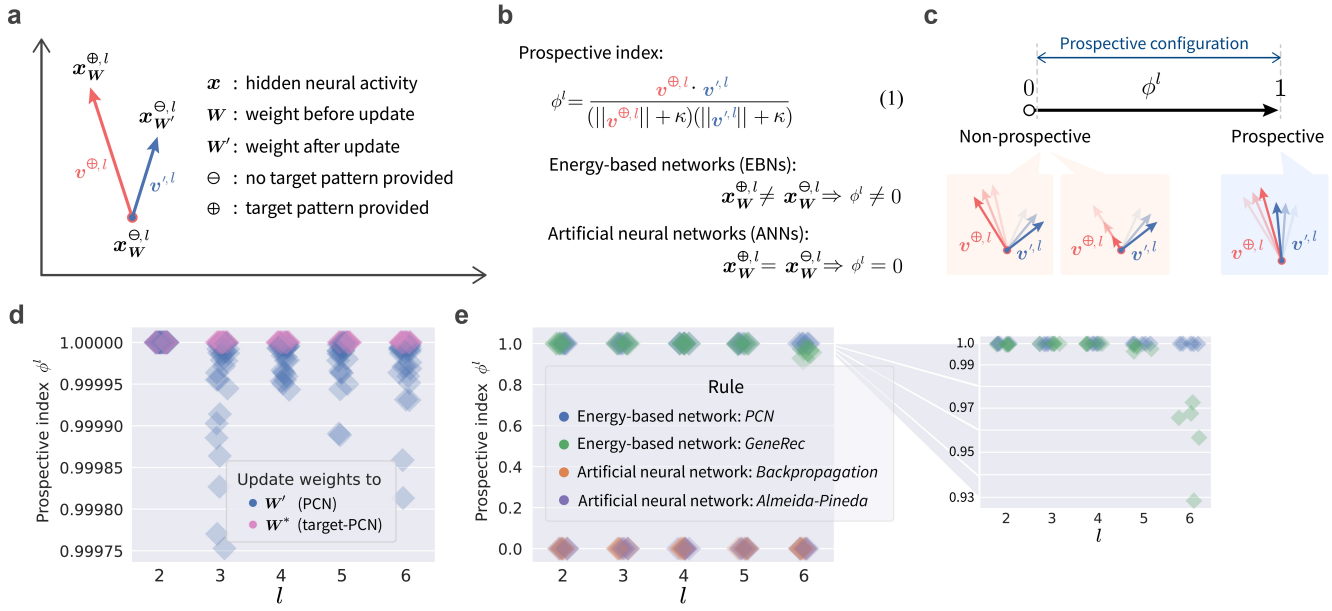
Competing interests

The authors declare no competing interests.

Additional information

Extended Data Figures/Tables is available for this paper in the same file (Section 1).
Supplementary Information is available for this paper in the same file (Section 2).
Correspondence and requests for materials should be addressed to Y.S. and R.B.

1 Extended Data



Extended Data Fig. 1

Formal definition of prospective configuration. Formal definition of prospective configuration with prospective index (panels a–c), a metric that one can measure for any learning model. With this metric, we show that prospective configuration is present in different *energy-based networks* (EBNs), but not in *artificial neural networks* (ANNs) (panels d–e).

► **a|** To introduce the prospective index, we consider the hidden neural activity $\boldsymbol{x}^l$ in layer l, in three moments of time. First, a learning iteration starts from $\boldsymbol{x}^l$ under the current weights $\boldsymbol{W}$ without target pattern provided $\ominus$: $\boldsymbol{x}_{\boldsymbol{W}}^{\ominus,l}$. Second, a target pattern is provided $\oplus$, and neural activity settles to $\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}$. Third, $\boldsymbol{W}$ is updated to $\boldsymbol{W}'$, the target pattern is removed $\ominus$, and the neural activity settles to $\boldsymbol{x}_{\boldsymbol{W}'}^{\ominus,l}$. We define two vectors $\boldsymbol{v}^{\oplus,l}$ and $\boldsymbol{v}'^{,l}$, representing the direction of the neural activity's changes as a result of the target pattern being given $\ominus \rightarrow \oplus$ and the weights being updated $\boldsymbol{W} \rightarrow \boldsymbol{W}'$, respectively.

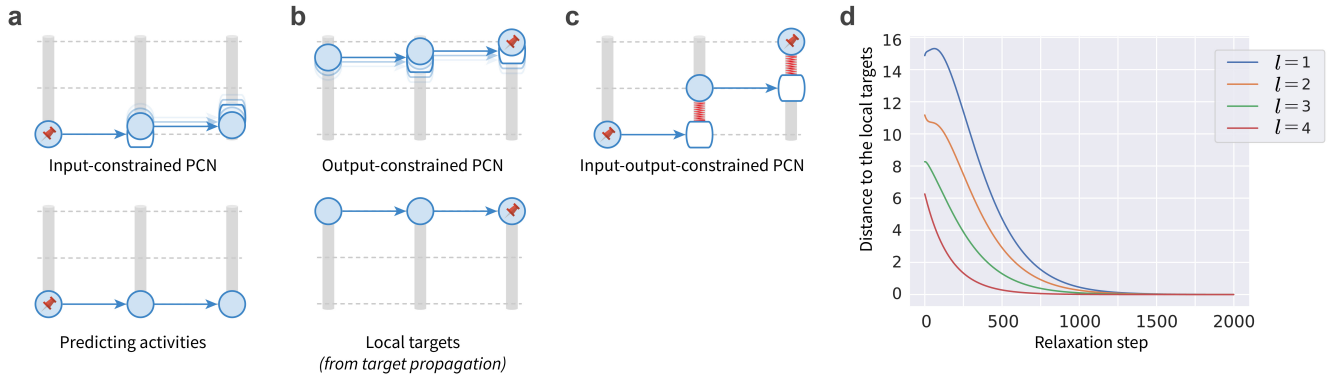
► **b|** The prospective index ϕ^l is the cosine similarity of $\boldsymbol{v}^{\oplus,l}$ and $\boldsymbol{v}'^{,l}$. A small constant $\kappa = 0.00001$ is added in the denominator to ensure that the prospective index is still defined if the length of one of the vectors is 0 (in which case the prospective index in equal to 0). For EBNs, the neural activity settles to a new configuration when the target pattern is provided, i.e., $\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l} \neq \boldsymbol{x}_{\boldsymbol{W}}^{\ominus,l}$, so ϕ^l is non-zero; for ANNs, the neural activity stays unchanged when the target pattern is provided, i.e., $\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l} = \boldsymbol{x}_{\boldsymbol{W}}^{\ominus,l}$, so ϕ^l is zero.

► **c|** A positive ϕ^l implies that $\boldsymbol{v}^{\oplus,l}$ and $\boldsymbol{v}'^{,l}$ are pointing in the same direction, i.e., the neural activity after the target pattern provided $\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}$ is similar to the neural activity after the weight update $\boldsymbol{x}_{\boldsymbol{W}'}^{\ominus,l}$, i.e., is prospective. We define the models following the principle of prospective configuration as those with positive ϕ^l (averaged over all layers). Additionally, prospective index close to 1 implies that a weight update rule in a model is able to consolidate the pattern of activity following relaxation, so a similar pattern is reinstated during prediction on the next trial.

► **d|** The prospective index ϕ^l of different layers l in PCNs and a variant of PCNs called target-PCNs. Several observations can be made, and they are explained and proved in SI 2.3.

► **e|** The prospective index ϕ^l of different EBNs and ANNs. Here, we can see that all EBNs produce positive ϕ^l, i.e., the prospective configuration is commonly observed in EBNs, but not in ANNs.

Implementation details. We train various models to predict a target pattern from an input pattern (both randomly generated from $\mathcal{N}(0,1)$ and of 32 entries). The structure of the networks is $32 \to 32 \to 32 \to 32 \to 32 \to 32 \to 32$. The weights are initialized using Xavier normal initialization[94] (described in the Methods). The models were trained for one iteration (i.e., one update of the weights), the prospective index was then measured for this update. Prospective indices of input and output layers are not reported. This is because the input and output layers are held fixed during learning; thus, the prospective index is not defined for them. Experiments were repeated 20 times. The EBNs investigated include PCNs[30,43,55], target-PCNs, and *GeneRec*[89], while the ANNs investigated include backpropagation and *Almeida-Pineda*[90–92] Details of all simulated models are given in Section 2.1 of Supplementary Information.



Extended Data Fig. 2

Relationship of prospective configuration to target propagation. Prospective configuration is related to another influential algorithm of credit assignment — target propagation[60]. Since target propagation has been closely linked to an efficient optimization method by Gauss-Newton[105,106], this relationship further provides another perspective on the advantages of prospective configuration. Target propagation is an algorithm, which explicitly computes the neural activity in hidden layers required to produce the desired target pattern. We call these values local targets. We demonstrate that one of energy-based networks, predictive coding networks[30,43,55] (PCNs) tends to move the activity during relaxation towards these local targets. This provides an additional intuition for why in the PCNs the neural activity converges to a prospective state: These local targets correspond to a prospective state, because if they were produced, they would predict the desired target pattern. The relationship of PCNs to target propagation can be visualized with the proposed energy machine in Fig. 2, hence panels a–c illustrate how the neural activity in a PCN depends on whether inputs and outputs are constrained, and these properties are formally proved in SI 2.4.

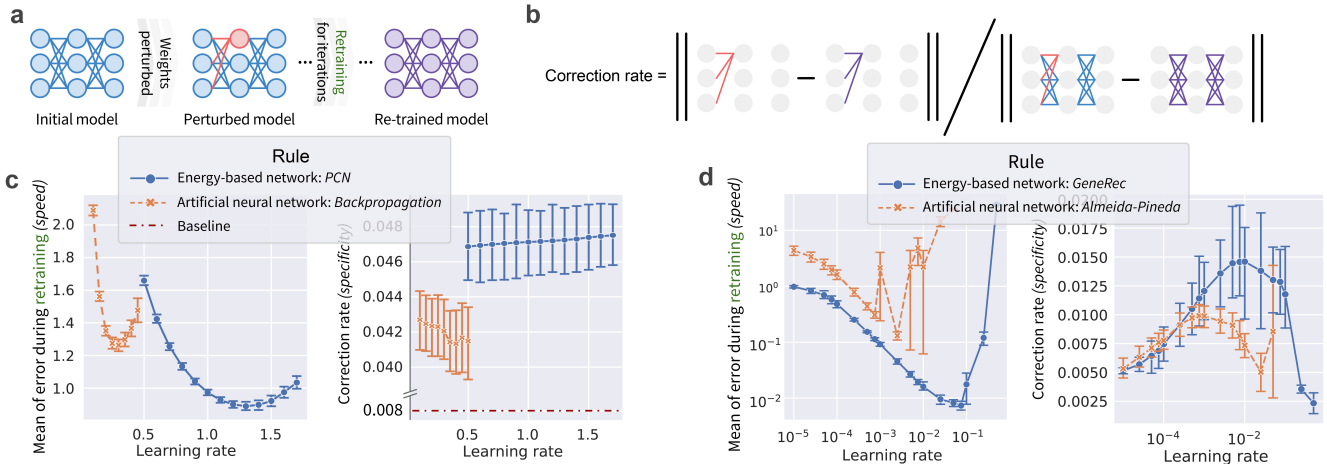
▶ **a**| With only input neurons constrained (and outputs unconstrained) PCNs can generate prediction about the output, and hence we refer to this pattern of neural activity as the predicting activity.

▶ **b**| With only output neurons constrained (and inputs unconstrained), the neural activity of PCNs relaxes to the local target from target propagation. This happens because with only outputs constrained, other nodes have a freedom to move to values that generate the outputs, and when the energy reduces to 0 (as shown in the bottom display) all neurons must have the activity generating the target output.

▶ **c**| With both input and output neurons constrained, the neural activity of PCNs relaxes to the weighted sum of the local target from target propagation and the predicting activity. Note that the position of the hidden node is in between the positions from panels a and b.

▶ **d**| The distance between the neural activity to the local target at different layers along the relaxation progress in output-constrained PCNs. Here, the neural activity of the output-constrained PCNs converges to the local target, and the layers closer to the output layer (larger l) converge to the local target earlier than the others, which is as expected from the physical intuition of the energy machine.

Implementation details. We train the models to predict a target pattern from an input pattern (both randomly generated from $\mathcal{N}(0,1)$, and the input and target patterns are of 5 and 1 entries, respectively). The structure of the networks is $5 \to 5 \to 5 \to 5 \to 1$. There is no activation function, i.e., it is a linear network. For the computation of the local target in target propagation, refer to the original paper[60]. The mean square difference is used to measure the distance to the local target.



Extended Data Fig. 3

Prospective configuration yields a more accurate weight modification. A numerical experiment (panels a–b) verifies that *energy-based networks* (EBNs) yield a accurate weight modification than *artificial neural networks* (ANNs) (panels c–d). The following intuition can be provided for why the prospective configuration enables an accurate weight modification. In EBNs, if more error is assigned to a neuron, this neuron will settle to a prospective activity that reduces the error. The prospective activity of this neuron is then propagated through the network, resulting in less error being assigned to other neurons, thus the error being assigned more accurately.

▶ **a|** Experimental procedure: we take a pre-trained model (illustration here does not reflect the real size of the model), randomly select a hidden neuron and perturb the synaptic weights connecting to this neuron (red), then retrain this model on the same pattern for a fixed number of iterations. During retraining, an optimal learning agent is expected to identify that the error in the output neurons is due to the perturbed weights, thus, (1) correct the error faster, and (2) correct the perturbed weights more. We refer to the above two properties as *speed* and *specificity*. Speed can be measured with the mean of error over retraining iterations (the lower, the better).

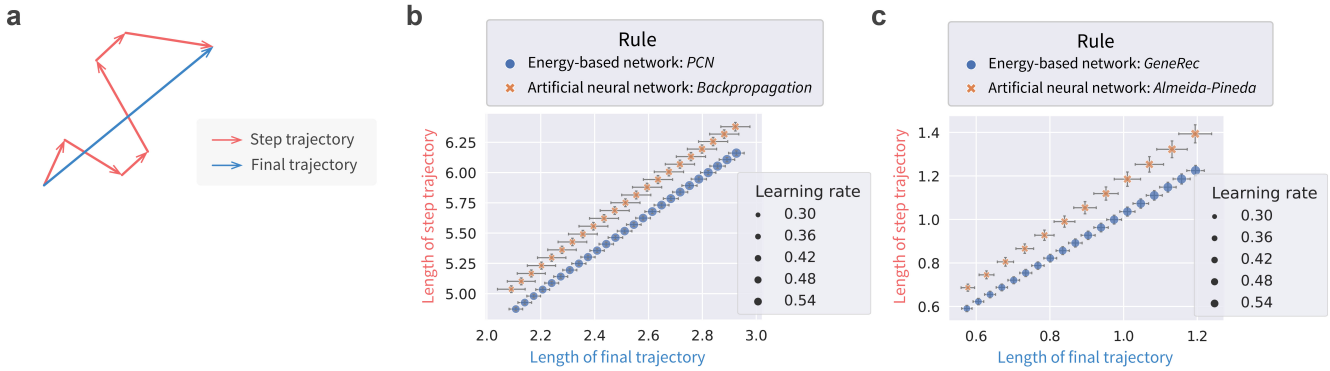
▶ **b|** Specificity can be measured by correction rate (the higher, the better): the ratio of how much the perturbed weights are corrected compared to how much all the weights (in all layers) are corrected after all retraining iterations.

▶ **c|** A comparison between an EBN, predictive coding network[30,43,55] (PCN), and an ANN, trained with backpropagation. In the right plot, there is an additional baseline, which is the number of perturbed weights divided by the number of all the weights, indicating the expected correction rate if a learning rule randomly assigns errors.

▶ **d|** The same comparison as in panel c, but for another EBN, namely, *GeneRec*[89]. GeneRec describes learning in recurrent networks, and ANN with this architecture is not trained by standard backpropagation, but by a variant of backpropagation, called *Almeida-Pineda*[90–92].

Implementation details. We first pre-train the models to predict a target pattern from an input pattern (both randomly generated from $\mathcal{N}(0,1)$ and of 32 entries). The structure of the networks is $32 \to 32 \to 32 \to 32$. The pre-training session is sufficiently long (1000 iterations) to reach convergence. Then, one neuron is randomly selected from the $(32+32)$ hidden neurons, and all weights connecting to this neuron are "flipped" (i.e., multiplied by -1). Current weights of the network are recorded as $\boldsymbol{W}_b$. The part of current weights that were just flipped are recorded as $\boldsymbol{W}_b^f$. The network is then re-trained on the same pattern for 64 iterations. After each re-training iteration, the model makes a prediction. The square difference

between the prediction and the target pattern is recorded as the "error during re-training" of this iteration. After the entire re-training session, the "errors during re-training" are averaged over the 64 re-training iterations, producing the left plots of panels c–d. Current weights of the network are recorded as $\boldsymbol{W}_a$. The part of current weights that were flipped before the re-training session are recorded as $\boldsymbol{W}_a^f$. The *correction rate* is computed as $\| \boldsymbol{W}_a^f - \boldsymbol{W}_b^f \| \ / \ \| \boldsymbol{W}_a - \boldsymbol{W}_b \|$, which produces the right plots of panels c–d. Each configuration was repeated 20 times, and the error bars represent standard error.



Extended Data Fig. 4

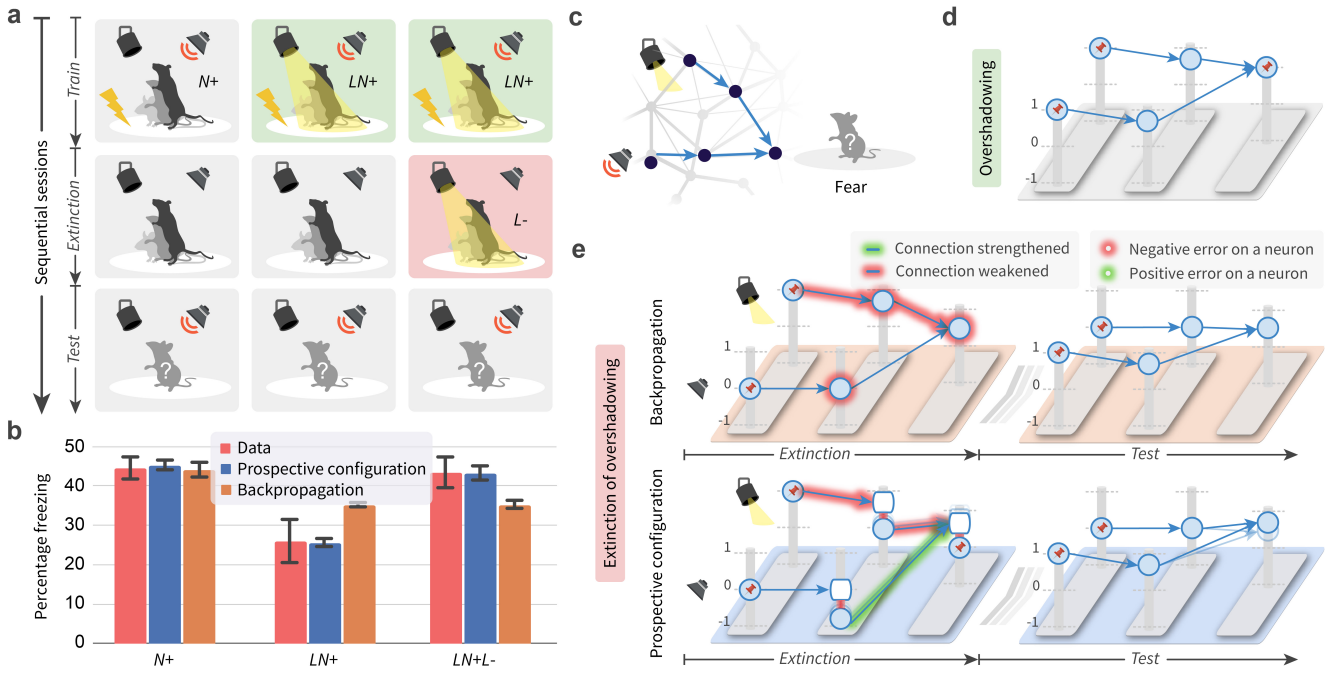
Prospective configuration produces less erratic weight modification. An experiment verifies that *energy-based networks* (EBNs) (i.e., prospective configuration), produce a less erratic weight modification than *artificial neural networks* (ANNs) (i.e., backpropagation).

▶ **a**| Experimental procedure. The weights are updated for a fixed number of steps on a fixed number of data points, which produces the step trajectory in the weight space (each red arrow corresponds to one weights update). Connecting the start and end points of the step trajectory (i.e., the initial and final weights of the model) produces the final trajectory (blue). A learning rule with less erratic weight modification would produce a shorter step trajectory relative to the final trajectory. This property of less erratic weight modification is also desirable for biological systems, because each weight modification costs metabolic energy.

▶ **b**| Comparison of the length of step and final trajectories between EBN, predictive coding network (PCN), and an ANN, trained with backpropagation. Note that the length of both trajectories depends on the learning rate. Thus, in panels b–c, we present the length of the step and final trajectory on y and x axis, respectively; each point is from a specific learning rate (represented by the size of the marker; the legend does not enumerate all sizes). In such plots, when the two learning rules produce roughly the same length of final trajectory (which could be from different learning rates), one can compare the length of their step trajectory.

▶ **c**| The same comparison as in panel c, but for another EBN, namely, *GeneRec*[89]. GeneRec describes learning in recurrent networks, and ANN with this architecture is not trained by standard backpropagation, but by a variant of backpropagation, called *Almeida-Pineda*[90–92].

Implementation details. We train the models to predict a target pattern from an input pattern (both randomly generated from $\mathcal{N}(0,1)$ and of 32 entries), and there are 32 pairs of them (32 datapoints). The structure of the networks is $32 \to 32 \to 32 \to 32$. The batch size is one, as biological systems update the weights after each experience. The training is conducted for 64 epochs (one epoch iterates over all 32 datapoints). At the end of each epoch, current weights of the network are recorded as one set. Thus, it results in a sequence of 64 sets of weights. Each set of weights is used as one point to construct the step trajectory. The first and last sets of weights are used to construct the final trajectory. The length of the step and final trajectories can then be computed and reported in Extended Data Fig. 4b–c. For each combination of learning rule and learning rate, simulation is repeated 20 times with different seeds, and the error bars represent standard error.



Extended Data Fig. 5

Prospective configuration explains extinction of overshadowing in fear conditioning (complete description of the experiment in Fig. 6). The extinction of overshadowing effect[78] can be accurately reproduced and explained by prospective configuration, but not backpropagation (comparing "Data" against "Prospective configuration" and "Backpropagation" in panel b).

► **a|** Experimental procedure. Rats were divided into three groups, corresponding to three columns. Each group underwent three sessions sequentially, corresponding to the top three rows, namely, train, extinction, and test. The goal of the training session was to associate fear (+) with different presented stimuli *N* or *LN* depending on the group: rats experienced an electric shock paired with different stimuli, where *N* and *L* stands for noise and light, respectively. Next, during the extinction session no shock was given, and for the third group the light was presented but without the shock, aiming to eliminate the fear (-) of light (*L*). Finally, all groups underwent a test session measuring how much fear was associated with the noise: the noise was presented and the percentage of freezing of rats was measured.

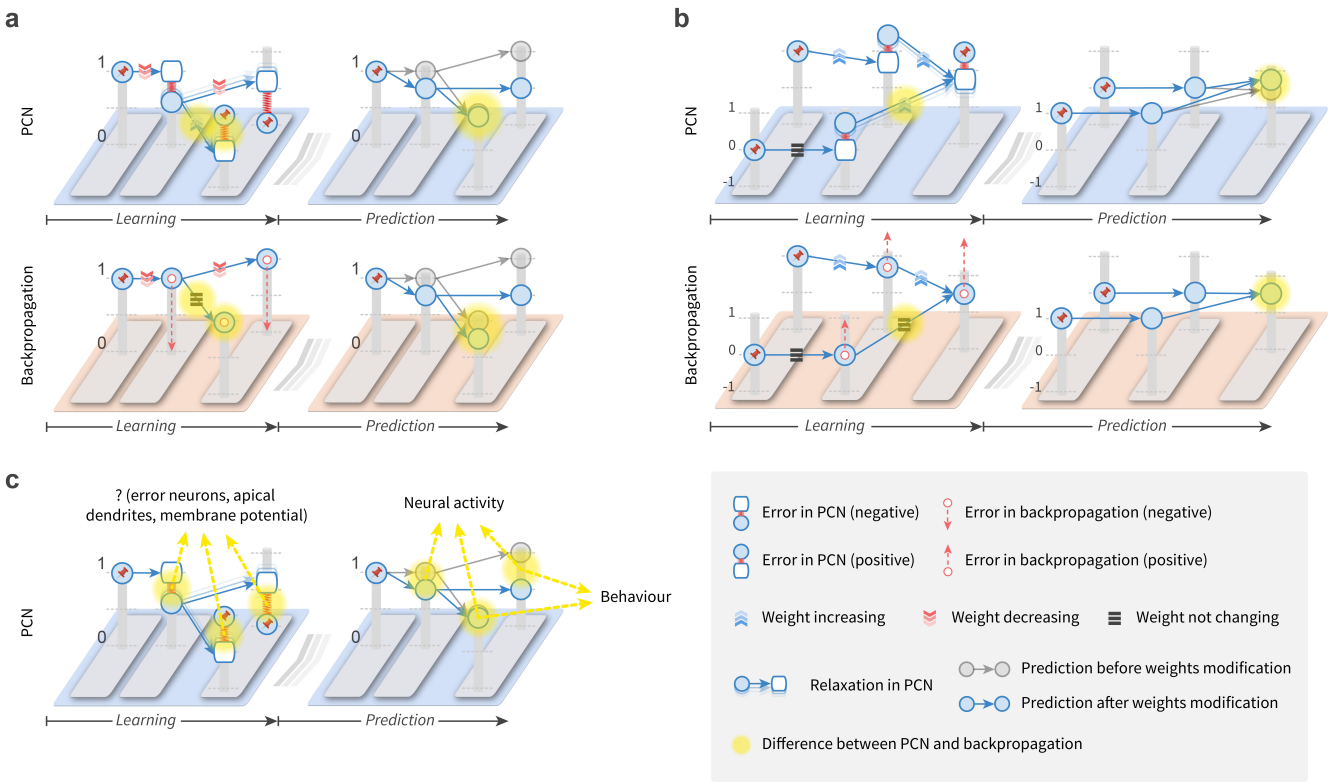
► **b|** Experimental and simulation results. The bar chart plots the percentage of freezing during test for each group, both measured in the animal experiments[78] (i.e., Data) and simulated by the two learning rules. Two effects are present in experimental data. First, comparing the groups *N*+ and *LN*+ demonstrates the overshadowing effect: there is less fear of noise if the noise had been compounded with light when paired with shock *LN*+ than if the noise alone had been paired with shock *N*+ (that is, light overshadows noise in a conditioned fear experiment). This effect can be accounted for by the canonical model of error-driven learning — the Rescorla-Wagner model[83], and consequently it can be also produced by both error-driven models we consider — backpropagation and prospective configuration (explained in panel d). Second, comparing the groups *LN*+ and *LN*+*L*- shows the striking effect of extinction of overshadowing: presenting the light without the shock increases the fear response to the non-presented stimulus — noise. This effect is not produced by backpropagation, but can be reproduced by prospective configuration (explained in panel e).

► **c|** The neural architecture considered: both stimuli are processed by hidden neurons (i.e., intermediate neurons corresponding to visual and auditory cortices) and are then combined to produce the prediction

of electric shock (i.e., fear).

► **d**| Explanation of overshadowing effect, i.e., the reduced percentage freezing comparing group *LN*+ against *N*+. With the energy machine introduced in Fig. 2, the diagram illustrates the state of the network after the *Train* sessions in groups *LN*+ and *LN*+*L*-. The network learns to predict a shock (i.e. produces output of 1), on the basis of two stimuli, hence each of the inputs to the output neuron must be 0.5. Therefore, if only one stimulus is presented, the output of the network is reduced to 0.5. The network shown in this panel is acting as the starting point of learning in panel e.

► **e**| Explanation of extinction of overshadowing effect, i.e., the increased percentage freezing after noise in group *LN*+*L*- in comparison to *LN*+. This effect suggests that during extinction trials, where light is presented without a shock, the animals increased fear prediction to noise. As shown in this panel, backpropagation (top) cannot explain this, since the error cannot be backpropagated to and drive a weight modification on a non-activated branch where no stimuli are presented; prospective configuration (bottom), however, can account for this. Specifically, on the non-activated branch, the hidden neural activity decreases from zero to a small negative value (it may correspond to a neural activity decreasing below the baseline[10]). Since a weight modification depends on the product of the presynaptic activity and the postsynaptic activity representing the error, which are both negative here, the weight on the non-activated branch is strengthened.



Extended Data Fig. 6

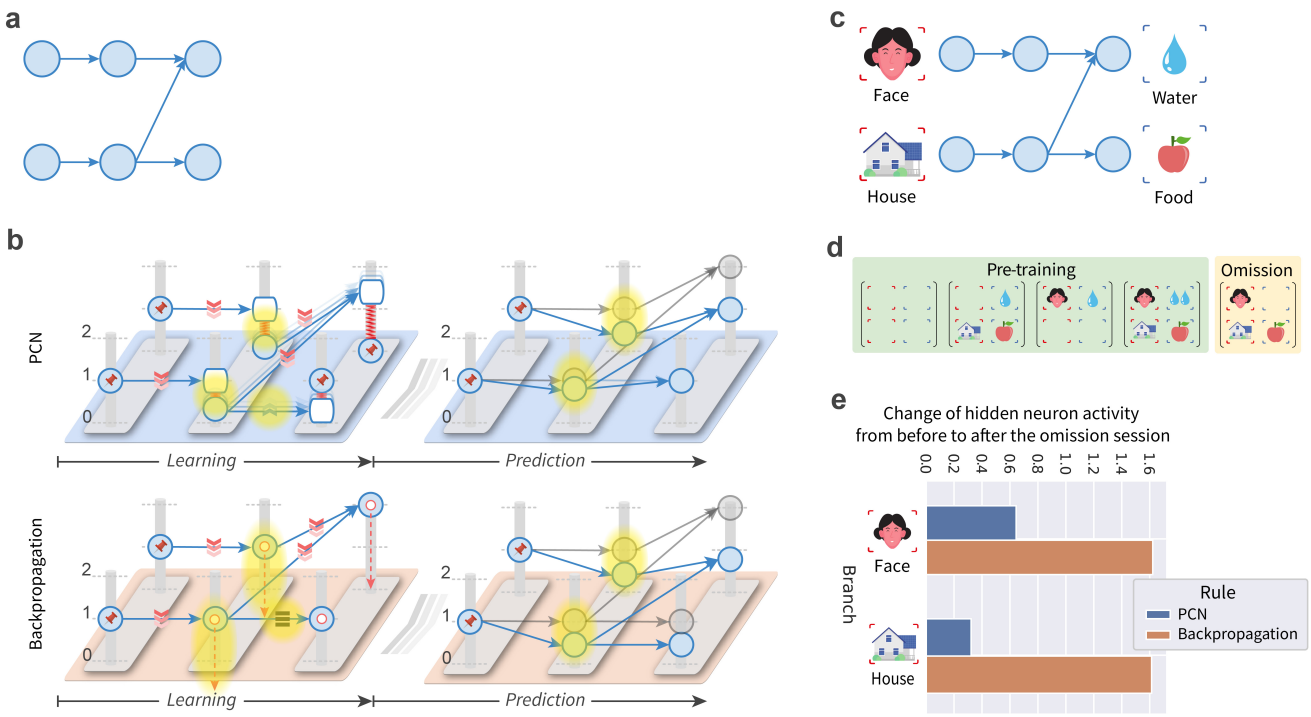
Experimental predictions of prospective configuration and backpropagation. To provide examples of experimental predictions of prospective configuration, panels a-b (and Extended Data Fig. 7) highlight the different behaviour of the learning rules in simple network motifs, which are minimal networks displaying given behaviour. Two motifs in this figure have been already analysed earlier in the paper, but there we focused on differences corresponding to experimentally observed effects, while in this figure we also highlight other qualitative differences that reveal a range of untested predictions of prospective configuration. Here, we consider *predictive coding network*[30,43,55] (PCN) with the energy machine in Fig. 2, however, a similar analysis can be applied to other energy-based networks, which also follow the principle of prospective configuration. In each panel, the top and bottom rows demonstrate the prediction of PCNs and backpropagation, respectively. The left column highlights the differences in the prediction errors during learning and the resulting weight update. The right column demonstrates the neuron activity before (transparent) and after (opaque) weight update. The differences between the rules are highlighted in yellow. Experimental predictions following from them can be derived as summarized in panel c.

► **a|** The error may spread to the branch where the prediction is correctly made. This motif has been compared with experimental data in Fig. 7, but here we focus on the effect illustrated in Fig. 1 and Fig. 2d, which despite being intuitive, has not been tested experimentally to our knowledge. The panel highlights that an error on one output in PCN results in prediction error on the other, correctly predicted output. This produces an increase of the weight of the correct output neuron, which compensates for the decrease of the weight from the input, and enables the network to make correct prediction on the next trial.

► **b|** The error may cause a weight change in the sensory regions associated with absent stimuli. The panel shows a similar motif as the one investigated in Extended Data Fig. 5. The difference is that Extended Data Fig. 5 introduces negative error while this panel introduces positive error on the same architecture. Interestingly, introducing negative (Extended Data Fig. 5) or positive (this panel) error to the

same architecture produce a similar effect in the PCN, i.e. an increased predicted output for the stimulus not presented during learning.

► **c|** Observing model behaviour in experiments. The diagram summarizes how the differences highlighted in previous panels could be measured in experiments. The key difference in models' behaviour during learning is the difference in error signals. However, currently it is not clear how the prediction errors are represented in the cortical circuits. Three hypotheses have been proposed in the literature that errors are encoded in: activity of separate error neurons[43,107,108], membrane potential of value neurons[109,110], membrane potential in apical dendrites of value neurons[3,27]. Nevertheless, if the future research establishes how errors are encoded, it will be possible to test the predictions related to errors during learning. For example, one can design a task corresponding to panel a, where predictions in two modalities have to be made on the basis of a stimulus. One can then test if omission in one modality results in error signals in the brain region corresponding to the correctly predicted modality. The models also differ in the neural activity of the value nodes during the next trial following the learning. Such predictions are easier to test, because if the model makes a prediction without observing any supervised signal, then all errors are equal to 0 in PCNs, so the neural activity should reflect just the activity of value nodes. Additionally, the differences in the activity of the output value neurons should be testable in behavioural experiments. For example, panel b makes a behavioural prediction (presenting light with stronger shock should also increase freezing for tone) that can be tested in a similar way as described in Extended Data Fig. 5. Testing this prediction would also validate our explanation of the experimental result in Extended Data Fig. 5.



Extended Data Fig. 7

Experimental predictions concerning errors assigned to hidden nodes. The figure demonstrates a striking difference in how prospective configuration and backpropagation assign error to hidden nodes. Namely, in prospective configuration, the error assigned to a hidden node is reduced if the node is also connected to correctly predicted outputs. This difference is illustrated in a motif (panel a), for which we illustrate behaviour of learning rules with the energy machine (panel b), and describe a sample experiment testing model predictions (panels c–d) . Finally, we report the simulation results of the two learning rules (panel e), confirming that they indeed make distinct predictions for this motif.

► **a**| In this motif, two stimuli are presented and two predictions are made. One stimulus contributes to only one prediction, while the other stimulus contributes to both predictions.

► **b**| Comparison of learning rules' behaviour with the energy machine (notation as in Extended Data Fig. 6). The diagrams illustrate a network containing the motif (panel a), in a situations where one of the predicted outputs (top output) is omitted. A negative error is introduced to the prediction determined by both stimuli. Thus, we would expect the error to be assigned to hidden neurons on both branches. Both learning rules do so, however, they assign errors differently. PCNs allocate less error on the bottom hidden neuron than the top hidden neuron, because the bottom hidden neuron also contributes to another output that was correctly predicted, while backpropagation assigns the same error to both hidden neurons. This is also a nice example where prospective configuration (PCNs) demonstrates more intelligent behavior.

► **c**| Experimental stimuli. To test this motif, it is important to choose stimuli for which neural activity of "hidden" neurons can be easily measured. In case of a human experiment, inputs could contain faces and houses, because the hidden neurons would correspond to the brain regions known to be specifically excited by these particular types of stimuli, and the activity of these regions could be easily distinguished in an experiment[111]. The outputs could correspond to reward modalities (e.g. water and food). In case of a human experiment, these could be "virtual rewards" the participants are instructed to gather, while for animals, these could be the actual rewards.

► **d**| Experimental procedure. The motif shown in panel c could arise in brain networks from training

with examples shown in the green box. To test differences in behaviour of learning rules, partial omission trials could be presented, in which one of the expected outputs is omitted, as shown in the orange box.

► **e|** Results of simulations. We pre-train the models with the examples in the green box in panel d for a sufficient number of iterations until convergence, and then we train the model with the omission using the example in the orange box in panel d for one trial. We measure the change of hidden neural activity on both branches from before to after the above omission session. The graph shows simulation results of such change in hidden activity: PCNs predict different changes on different branches, while backpropagation predicts the same change on different branches (consistent with illustration in panel b, right).

Implementation details. Presenting and not presenting a stimulus (face, house, water, or food) are encoded as 1 and 0, respectively. Presenting two drops of water is encoded as 2. The network is initialized to the pre-trained connection pattern demonstrated in Extended Data Fig. 7c, i.e., the weights visible on the panel are set to one and other weights are set to zero. Such pattern of weights would arise from pre-training with the four examples in Extended Data Fig. 7d (in the green "Pre-training" box), but for simplicity, we do not simulate such pre-training but just set the weights as explained before. Next, to measure the activity of hidden units of such network during prediction, we set both inputs to 1 and record the hidden neural activity of the two branches. Subsequently, the model is presented with the omission trial shown in the orange box and the weights are updated once. Finally, to measure weight changes resulting from training on the subsequent prediction trial, we set both inputs to 1 and record the hidden neural activity of the two branches for the second time. The change of the hidden neuron activity from before to after the omission session can thus be computed for both branches.

2 Supplementary Information

In this supplement, we present additional description and analysis of the simulated models. The first part provides details of all models simulated in the paper. The second part analyses target alignment and the third part - prospective index of PCNs. The fourth part discusses relationship between prospective configuration and target propagation.

2.1 Details of simulated models

This section gives more details of all simulated models. The general idea of *energy-based networks* (EBNs) and *artificial neural networks* (ANNs), and one of EBNs, *predictive coding network*[30,43,55] (PCN), have been described in the Main text and Methods. PCN is again included here along with other simulated models to provide descriptions in a unified form, facilitating the reproduction of our reported results. Complete code and full documentation reproducing all simulation results will be made publicly available at `https://github.com/YuhangSong/A-New-Perspective` upon publication of this work.

Algorithms 2 to 7 describe how the four models simulated in this paper predict and learn. These four models are: PCN, backpropagation, *GeneRec*[89], and *Almeida-Pineda*[90–92]. Among the four models, PCN and GeneRec are the two EBNs we investigate; backpropagation and Almeida-Pineda are the two ANNs we investigate. Specifically, PCN is compared against backpropagation, because it has been established that PCN are closely related to backpropagation[30,37] and they make the same prediction with the same weights and input pattern[30]. Therefore we simulated prediction in these two algorithms in the same way (Algorithm 2). However, they learn differently (c.f. Algorithms 3 and 4). The other EBN, GeneRec, describes learning in recurrent networks, and ANN in this architecture is not trained by standard backpropagation, but a modified version proposed by Almeida and Pineda[90–92] (thus called the *Almeida-Pineda* algorithm). Thus, GeneRec should be compared against Almeida-Pineda because they make same prediction with the same weights and input pattern[89]. Therefore we simulated prediction in these two algorithms in the same way (Algorithm 5). But they learn differently (c.f. Algorithms 6 and 7). In a word, PCN and backpropagation are EBN and ANN working in feed-forward architecture, respectively; GeneRec and Almeida-Pineda are EBN and ANN working in recurrent architecture, respectively.

Algorithm 2: Predict with backpropagation or *predictive coding network*[30,43,55] (PCN)

Input: input pattern $\boldsymbol{s}^{\text{in}}$; synaptic weights $\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\}$
Result: activity of output neurons $\boldsymbol{x}^{L+1}$

$\boldsymbol{x}^1 = \boldsymbol{s}^{\text{in}}$; // Clamp input neurons to input pattern
for $l = 1; l < L+1; l = l+1$ **do** // Forward pass of the network
  $\boldsymbol{x}^{l+1} = \boldsymbol{w}^l f\left(\boldsymbol{x}^l\right)$;
end

Particularly, PCN & Backpropagation work in a network where prediction is made from the input through a series of forward weights $\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\}$; GeneRec & Almeida-Pineda works in a network where prediction is made from input through a mixture of forward weights $\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\}$ and backward weights $\{\boldsymbol{m}^1, \boldsymbol{m}^2, \cdots, \boldsymbol{m}^L\}$. The forward weights $\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\}$ and backward weights $\{\boldsymbol{m}^1, \boldsymbol{m}^2, \cdots, \boldsymbol{m}^L\}$ are not necessarily related. This architecture is also similar to the continuous Hopfield model[112,113]. Unlike in some previous studies[29], here, we focus on layered networks, where the sets of neurons at adjacent layers $\boldsymbol{x}^l$ and $\boldsymbol{x}^{l+1}$ are connected by synaptic weights. Thus, we define two sets of weights for GeneRec & Almeida-Pineda that works in the recurrent network: $\boldsymbol{w}^l$ is the forward weights connecting from $\boldsymbol{x}^l$ to $\boldsymbol{x}^{l+1}$; $\boldsymbol{m}^l$ is the backward weights connecting from $\boldsymbol{x}^{l+1}$ to $\boldsymbol{x}^l$.

Algorithm 3: Learn with backpropagation

Input: input pattern $\boldsymbol{s}^{\text{in}}$; target pattern $\boldsymbol{s}^{\text{target}}$; synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$
Output: updated synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$

1 $\boldsymbol{x}^1 = \boldsymbol{s}^{\text{in}}$; // Clamp input neurons to input pattern
2 **for** $l = 1;\ l < L+1;\ l = l+1$ **do** // Forward pass of the network
3 $\quad \boldsymbol{x}^{l+1} = \boldsymbol{w}^l f\left(\boldsymbol{x}^l\right)$;
4 **end**
5 $\boldsymbol{\varepsilon}^{L+1} = \boldsymbol{s}^{\text{target}} - \boldsymbol{x}^{L+1}$; // Compute error of the output neurons
6 **for** $l = L+1;\ l > 2;\ l = l-1$ **do** // Backpropagation of error
7 $\quad \boldsymbol{\varepsilon}^{l-1} = f'\left(\boldsymbol{x}^{l-1}\right) \circ \left(\left(\boldsymbol{w}^{l-1}\right)^T \boldsymbol{\varepsilon}^l\right)$;
8 **end**
9 **for** $l = 1;\ l < L+1;\ l = l+1$ **do** // Update weights
10 $\quad \Delta \boldsymbol{w}^l = \alpha \boldsymbol{\varepsilon}^{l+1} \left(f\left(\boldsymbol{x}^l\right)\right)^T$;
11 $\quad \boldsymbol{w}^l = \boldsymbol{w}^l + \Delta \boldsymbol{w}^l$;
12 **end**

Also note that GeneRec has been explored and re-discovered in recent works[50,114] showing how a closely related algorithm resembles backpropagation when the backward weights are the transposes of the forward weights $\boldsymbol{m}^l = \left(\boldsymbol{w}^l\right)^T$ (or for a fully-connected network in their context $w_{i,j} = w_{j,i}$), and how the extreme version of the algorithm approximate backpropagation[29].

Some common notations in the algorithms are: α is the learning rate for weights update; γ and $\mathscr{T}$ are the integration step and length of relaxation, respectively (specified to the two EBNs, PCN and GeneRec); $\boldsymbol{s}^{\text{in}}$ and $\boldsymbol{s}^{\text{target}}$ are the input and target patterns, respectively. For Almeida-Pineda, which requires additional iterative process to propagate error, β and $\mathscr{K}$ are the integration step and length of this iterative process, respectively. In our simulation, we use $\beta = 0.01$ and $\mathscr{K} = 1600$.

All simulated models work in mini-batch mode, that is to say, one iteration is to update the weights for one step on a mini-batch of data randomly sampled from the training set for classification tasks. The above sampling is without replacement, i.e., the same examples will not be sampled again before the completion of a epoch, which is when the entire training set has been sampled once. For example, considering a dataset of 1000 examples with a batch-size (number of examples in a mini-batch) of 10, then each iteration would update weights for one step on 10 examples, and it will take 100 such iterations to complete one epoch. To implement the Algorithms 2 to 7 described below in mini-batch mode, one can simply add an extra-dimension, the size of which is batch-size, to all the neuron-specific vectors in the algorithms such as $\boldsymbol{x}^l$, $\boldsymbol{\varepsilon}^l$ and etc., and then reduce this dimension by summing over it when computing weight update $\Delta \boldsymbol{w}^l$ (and $\Delta \boldsymbol{m}^l$ if the model is GeneRec or Almeida-Pineda).

Note that learning with Almeida-Pineda involves relaxation of the model, i.e., updating neural activity, in lines 5-12 of Algorithm 6. However, its function is to make a prediction with current weights and input pattern so that the error on the output neurons can be computed (in the following line 13), similar as the function of "forward pass" in backpropagation in lines 2-4 of Algorithm 3. The neural activity in the Almeida-Pineda model is fixed during spreading of error, like in backpropagation. Thus, Almeida-Pineda is classified as an ANN rather than an EBN (which updates neural activity during spreading of error).

Algorithm 4: Learn with *predictive coding network*[30,43,55] (PCN)

Input: input pattern $\boldsymbol{s}^{\text{in}}$; target pattern $\boldsymbol{s}^{\text{target}}$; synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$

Output: updated synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$

$\boldsymbol{x}^1 = \boldsymbol{s}^{\text{in}}$; // Clamp input neurons to input pattern

$\boldsymbol{x}^{L+1} = \boldsymbol{s}^{\text{target}}$; // Clamp output neurons to target pattern

for $l = 2;\ l < L+1;\ l = l+1$ **do** // Initialize $\boldsymbol{x}$

  $\boldsymbol{x}^l = \boldsymbol{0}$;

end

for $t = 0;\ t < \mathscr{T};\ t = t+1$ **do** // Relaxation

  for $l = 1;\ l < L+1;\ l = l+1$ **do**

    $\hat{\boldsymbol{x}}^{l+1} = \boldsymbol{w}^l f\left(\boldsymbol{x}^l\right)$;

    $\boldsymbol{\varepsilon}^{l+1} = \boldsymbol{x}^{l+1} - \hat{\boldsymbol{x}}^{l+1}$;

  end

  for $l = 2;\ l < L+1;\ l = l+1$ **do**

    $\Delta \boldsymbol{x}^l = \gamma\left(-\boldsymbol{\varepsilon}^l + f'\left(\boldsymbol{x}^l\right) \circ \left(\left(\boldsymbol{w}^l\right)^T \boldsymbol{\varepsilon}^{l+1}\right)\right)$;

    $\boldsymbol{x}^l = \boldsymbol{x}^l + \Delta \boldsymbol{x}^l$;

  end

end

for $l = 1;\ l < L+1;\ l = l+1$ **do** // Update weights

  $\Delta \boldsymbol{w}^l = \alpha \boldsymbol{\varepsilon}^{l+1}\left(f\left(\boldsymbol{x}^l\right)\right)^T$;

  $\boldsymbol{w}^l = \boldsymbol{w}^l + \Delta \boldsymbol{w}^l$;

end

Algorithm 5: Predict with *Almeida-Pineda*[90–92] or *GeneRec*[89]

Input: input pattern $\boldsymbol{s}^{\text{in}}$; forward and backward synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$ and $\left\{\boldsymbol{m}^1, \boldsymbol{m}^2, \cdots, \boldsymbol{m}^L\right\}$

Result: activity of output neurons $\boldsymbol{x}^{L+1}$

$\boldsymbol{x}^1 = \boldsymbol{s}^{\text{in}}$; // Clamp input neurons to input pattern

for $l = 2;\ l < L+2;\ l = l+1$ **do** // Initialize $\boldsymbol{x}$

  $\boldsymbol{x}^l = \boldsymbol{0}$;

end

for $t = 0;\ t < \mathscr{T};\ t = t+1$ **do** // Relaxation

  for $l = 2;\ l < L+1;\ l = l+1$ **do**

    $\Delta \boldsymbol{x}^l = \gamma\left(-\boldsymbol{x}^l + \boldsymbol{m}^l f'\left(\boldsymbol{x}^{l+1}\right) + \boldsymbol{w}^{l-1} f'\left(\boldsymbol{x}^{l-1}\right)\right)$;

    $\boldsymbol{x}^l = \boldsymbol{x}^l + \Delta \boldsymbol{x}^l$;

  end

  $\Delta \boldsymbol{x}^{L+1} = \gamma\left(-\boldsymbol{x}^{L+1} + \boldsymbol{w}^L f'\left(\boldsymbol{x}^L\right)\right)$;

  $\boldsymbol{x}^{L+1} = \boldsymbol{x}^{L+1} + \Delta \boldsymbol{x}^{L+1}$;

end

Algorithm 6: Learn with *Almeida-Pineda*[90–92]

Input: input pattern $\boldsymbol{s}^{\text{in}}$; target pattern $\boldsymbol{s}^{\text{target}}$; forward and backward synaptic weights $\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\}$ and $\{\boldsymbol{m}^1, \boldsymbol{m}^2, \cdots, \boldsymbol{m}^L\}$

Output: updated forward and backward synaptic weights $\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\}$ and $\{\boldsymbol{m}^1, \boldsymbol{m}^2, \cdots, \boldsymbol{m}^L\}$

$\boldsymbol{x}^1 = \boldsymbol{s}^{\text{in}}$; // Clamp input neurons to input pattern

for $l = 2;\ l < L+2;\ l = l+1$ **do** // Initialize $\boldsymbol{x}$

  $\boldsymbol{x}^l = \boldsymbol{0}$;

end

for $t = 0;\ t < \mathscr{T};\ t = t+1$ **do** // Relaxation

  for $l = 2;\ l < L+1;\ l = l+1$ **do**

    $\Delta \boldsymbol{x}^l = \gamma\left(-\boldsymbol{x}^l + \boldsymbol{m}^l f'\left(\boldsymbol{x}^{l+1}\right) + \boldsymbol{w}^{l-1} f'\left(\boldsymbol{x}^{l-1}\right)\right)$;

    $\boldsymbol{x}^l = \boldsymbol{x}^l + \Delta \boldsymbol{x}^l$;

  end

  $\Delta \boldsymbol{x}^{L+1} = \gamma\left(-\boldsymbol{x}^{L+1} + \boldsymbol{w}^L f'\left(\boldsymbol{x}^L\right)\right)$;

  $\boldsymbol{x}^{L+1} = \boldsymbol{x}^{L+1} + \Delta \boldsymbol{x}^{L+1}$;

end

$\boldsymbol{\varepsilon}^{L+1} = \boldsymbol{s}^{\text{target}} - \boldsymbol{x}^{L+1}$; // Compute error of the output neurons

for $l = 1;\ l < L+1;\ l = l+1$ **do** // Initialize $\boldsymbol{\varepsilon}$

  $\boldsymbol{\varepsilon}^l = \boldsymbol{0}$;

end

for $t = 1;\ t < \mathscr{K}+1;\ t = t+1$ **do** // Backpropagation of error

  for $l = 2;\ l < L+1;\ l = l+1$ **do**

    $\Delta \boldsymbol{\varepsilon}^l = \beta\left(-\boldsymbol{\varepsilon}^l + f'\left(\boldsymbol{x}^l\right) \circ \left(\boldsymbol{m}^l \boldsymbol{\varepsilon}^{l+1}\right) + f'\left(\boldsymbol{x}^l\right) \circ \left(\boldsymbol{w}^{l-1} \boldsymbol{\varepsilon}^{l-1}\right)\right)$;

    $\boldsymbol{\varepsilon}^l = \boldsymbol{\varepsilon}^l + \Delta \boldsymbol{\varepsilon}^l$;

  end

  $\Delta \boldsymbol{\varepsilon}^1 = \beta\left(-\boldsymbol{\varepsilon}^1 + f'\left(\boldsymbol{x}^1\right) \circ \left(\boldsymbol{m}^1 \boldsymbol{\varepsilon}^2\right)\right)$;

  $\boldsymbol{\varepsilon}^1 = \boldsymbol{\varepsilon}^1 + \Delta \boldsymbol{\varepsilon}^1$;

end

for $l = 1;\ l < L+1;\ l = l+1$ **do** // Update weights

  $\Delta \boldsymbol{w}^l = \alpha \boldsymbol{\varepsilon}^{l+1} \left(f\left(\boldsymbol{x}^l\right)\right)^T$;

  $\boldsymbol{w}^l = \boldsymbol{w}^l + \Delta \boldsymbol{w}^l$;

  $\Delta \boldsymbol{m}^l = \alpha \boldsymbol{\varepsilon}^l \left(f\left(\boldsymbol{x}^{l+1}\right)\right)^T$;

  $\boldsymbol{m}^l = \boldsymbol{m}^l + \Delta \boldsymbol{m}^l$;

end

Algorithm 7: Learn with *GeneRec*[89]

Input: input pattern $\boldsymbol{s}^{\text{in}}$; target pattern $\boldsymbol{s}^{\text{target}}$; forward and backward synaptic weights $\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\}$ and $\{\boldsymbol{m}^1, \boldsymbol{m}^2, \cdots, \boldsymbol{m}^L\}$

Output: updated forward and backward synaptic weights $\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\}$ and $\{\boldsymbol{m}^1, \boldsymbol{m}^2, \cdots, \boldsymbol{m}^L\}$

$\boldsymbol{x}^1 = \boldsymbol{s}^{\text{in}}$; // Clamp input neurons to input pattern
for $l = 2;\ l < L+2;\ l = l+1$ **do** // Initialize $\boldsymbol{x}$
  $\boldsymbol{x}^l = \boldsymbol{0}$;
end
for $t = 0;\ t < \mathscr{T};\ t = t+1$ **do** // Relaxation
  for $l = 2;\ l < L+1;\ l = l+1$ **do**
    $\Delta \boldsymbol{x}^l = \gamma\left(-\boldsymbol{x}^l + \boldsymbol{m}^l f'\left(\boldsymbol{x}^{l+1}\right) + \boldsymbol{w}^{l-1} f'\left(\boldsymbol{x}^{l-1}\right)\right)$;
    $\boldsymbol{x}^l = \boldsymbol{x}^l + \Delta \boldsymbol{x}^l$;
  end
  $\Delta \boldsymbol{x}^{L+1} = \gamma\left(-\boldsymbol{x}^{L+1} + \boldsymbol{w}^L f'\left(\boldsymbol{x}^L\right)\right)$;
  $\boldsymbol{x}^{L+1} = \boldsymbol{x}^{L+1} + \Delta \boldsymbol{x}^{L+1}$;
end
for $l = 1;\ l < L+1;\ l = l+1$ **do** // Update weights (negative phase)
  $\Delta \boldsymbol{w}^l = -\alpha f\left(\boldsymbol{x}^{l+1}\right)\left(f\left(\boldsymbol{x}^l\right)\right)^T$;
  $\boldsymbol{w}^l = \boldsymbol{w}^l + \Delta \boldsymbol{w}^l$;
  $\Delta \boldsymbol{m}^l = -\alpha f\left(\boldsymbol{x}^l\right)\left(f\left(\boldsymbol{x}^{l+1}\right)\right)^T$;
  $\boldsymbol{m}^l = \boldsymbol{m}^l + \Delta \boldsymbol{m}^l$;
end
$\boldsymbol{x}^{L+1} = \boldsymbol{s}^{\text{target}}$; // Clamp output neurons to target pattern
for $t = 0;\ t < \mathscr{T};\ t = t+1$ **do** // Relaxation
  for $l = 2;\ l < L+1;\ l = l+1$ **do**
    $\Delta \boldsymbol{x}^l = \gamma\left(-\boldsymbol{x}^l + \boldsymbol{m}^l f'\left(\boldsymbol{x}^{l+1}\right) + \boldsymbol{w}^{l-1} f'\left(\boldsymbol{x}^{l-1}\right)\right)$;
    $\boldsymbol{x}^l = \boldsymbol{x}^l + \Delta \boldsymbol{x}^l$;
  end
end
for $l = 1;\ l < L+1;\ l = l+1$ **do** // Update weights (positive phase)
  $\Delta \boldsymbol{w}^l = \alpha f\left(\boldsymbol{x}^{l+1}\right)\left(f\left(\boldsymbol{x}^l\right)\right)^T$;
  $\boldsymbol{w}^l = \boldsymbol{w}^l + \Delta \boldsymbol{w}^l$;
  $\Delta \boldsymbol{m}^l = \alpha f\left(\boldsymbol{x}^l\right)\left(f\left(\boldsymbol{x}^{l+1}\right)\right)^T$;
  $\boldsymbol{m}^l = \boldsymbol{m}^l + \Delta \boldsymbol{m}^l$;
end

2.2 Target alignment for networks without hidden layers (Fig. 3d)

Fig. 3d shows that target alignment for models without hidden layers, trained either with PC or BP, is exactly one, and here we prove this property analytically. Without hidden layers, PC and BP are identical algorithms. In a linear network, the change of the weight $\boldsymbol{w}^1$ is:

$$\Delta \boldsymbol{w}^1 \sim \boldsymbol{\varepsilon}^2 \left(\boldsymbol{x}^1\right)^T \tag{16}$$

We denote output after learning by $\boldsymbol{x}'^2$. The change of the output $\boldsymbol{x}'^2 - \boldsymbol{x}^2$ is:

$$\begin{aligned} \boldsymbol{x}'^2 - \boldsymbol{x}^2 &= \boldsymbol{w}'^{,2}\boldsymbol{x}^1 - \boldsymbol{w}^2\boldsymbol{x}^1 && (17)\\ &= \Delta\boldsymbol{w}^1\boldsymbol{x}^1 && (18)\\ &\sim \boldsymbol{\varepsilon}^2 \left(\boldsymbol{x}^1\right)^T \boldsymbol{x}^1 && (19) \end{aligned}$$

Here $\left(\boldsymbol{x}^1\right)^T \boldsymbol{x}^1$ is a positive scalar (if at least one entry in $\boldsymbol{x}^1$ is non-zero). Thus,

$$\boldsymbol{x}'^2 - \boldsymbol{x}^2 \sim \boldsymbol{\varepsilon}^2 \quad (20)$$

According to the definition of target alignment, which is the cosine similarity of the direction of the target (i.e., $\boldsymbol{\varepsilon}^2$) and the direction of learning (i.e., $\boldsymbol{x}'^2 - \boldsymbol{x}^2$), target alignment of this network is exactly one. This conclusion also applies to network with nonlinear activation function.

2.3 Formal proofs of prospective index of PCN (Extended Data Fig. 1)

This section formally proves two properties of the prospective index ϕ^l of *predictive coding network*[30,43,55] (PCN), that can be observed in Extended Data Fig. 1d. To briefly recap, prospective index ϕ^l quantifies to what extent the hidden neural activity of the network following clamping output neurons to a target pattern is shifting toward the hidden neural activity following subsequent weight modification. Below we show two properties visible in Extended Data Fig. 1d. Firstly, prospective index of the first hidden layer (ϕ^2) in a PCN is always one. Secondly, the prospective index in other layer is close to one because, the weights $\boldsymbol{W}$ in PCN are updated towards a configuration $\boldsymbol{W}^*$ whose prospective index is one.

2.3.1 Prospective index of the first hidden layer of PCN is always one

We assume that the model does not make a perfect prediction with the current weights, so that the error in the prediction drives the learning. As defined in Extended Data Fig. 1a, vectors $\boldsymbol{v}^{\oplus,l}$ and $\boldsymbol{v}'^{,l}$ describe the changes in hidden neuron activity, due to target pattern being provided and learning respectively. Specifically for layer $l = 2$, these vectors are:

$$\boldsymbol{v}^{\oplus,2} = \boldsymbol{x}_{\boldsymbol{W}}^{\oplus,2} - \boldsymbol{x}_{\boldsymbol{W}}^{\ominus,2} \quad (21)$$

$$\boldsymbol{v}'^{,2} = \boldsymbol{x}_{\boldsymbol{W}'}^{\ominus,2} - \boldsymbol{x}_{\boldsymbol{W}}^{\ominus,2} \quad (22)$$

We will now show that for PCN the above vectors $\boldsymbol{v}^{\oplus,2}$ and $\boldsymbol{v}'^{,2}$ point in the same direction. The change in activity due to learning $\boldsymbol{v}'^{,2}$ is equal to

$$\boldsymbol{v}'^{,2} = \boldsymbol{w}'^{,1} f\left(\boldsymbol{x}_{\boldsymbol{W}'}^{\ominus,1}\right) - \boldsymbol{w}^1 f\left(\boldsymbol{x}_{\boldsymbol{W}}^{\ominus,1}\right) \quad (23)$$

Since the value nodes of the first (input) layer $\boldsymbol{x}^1$ are always fixed to the input signal $\boldsymbol{s}^{\text{in}}$, the above Eq. (23) can further be written as,

$$\begin{aligned} \boldsymbol{v}'^{,2} &= \boldsymbol{w}'^{,1} f\left(\boldsymbol{s}^{\text{in}}\right) - \boldsymbol{w}^1 f\left(\boldsymbol{s}^{\text{in}}\right)\\ &= \left(\boldsymbol{w}'^{,1} - \boldsymbol{w}^1\right) f\left(\boldsymbol{s}^{\text{in}}\right)\\ &= \Delta\boldsymbol{w}^1 f\left(\boldsymbol{s}^{\text{in}}\right) && (24) \end{aligned}$$

Using Eqs. (13) and (11), we write

$$\boldsymbol{v}'^{,2} = \alpha \left(\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,2} - \hat{\boldsymbol{x}}_{\boldsymbol{W}}^{\oplus,2} \right) \left(f\left(\boldsymbol{s}^{\text{in}}\right) \right)^T f\left(\boldsymbol{s}^{\text{in}}\right) \tag{25}$$

In Eq. (25), $\hat{\boldsymbol{x}}^l$ denotes inputs to neurons in layer l, i.e., $\hat{\boldsymbol{x}}^l = \boldsymbol{w}^{l-1} f\left(\boldsymbol{x}^{l-1}\right)$. Note that $\hat{\boldsymbol{x}}_{\boldsymbol{W}}^{\oplus,2} = \boldsymbol{x}_{\boldsymbol{W}}^{\ominus,2}$, because both of these quantities are equal to $\boldsymbol{w}^1 f\left(\boldsymbol{s}^{\text{in}}\right)$ (the input of the first hidden layer ($l = 2$) does not change in response to output neuron being clamped). Using $\hat{\boldsymbol{x}}_{\boldsymbol{W}}^{\oplus,2} = \boldsymbol{x}_{\boldsymbol{W}}^{\ominus,2}$, the above Eq. (25) can further be written as,

$$\boldsymbol{v}'^{,2} = \left(\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,2} - \boldsymbol{x}_{\boldsymbol{W}}^{\ominus,2} \right) \alpha \left(f\left(\boldsymbol{s}^{\text{in}}\right) \right)^T f\left(\boldsymbol{s}^{\text{in}}\right) \tag{26}$$

Note that $\alpha \left(f\left(\boldsymbol{s}^{\text{in}}\right) \right)^T f\left(\boldsymbol{s}^{\text{in}}\right)$ is a positive scalar (if at least one entry in the input pattern is non-zero). Comparing Eqs. (21) and (26), we can see that vectors $\boldsymbol{v}'^{,2}$ and $\boldsymbol{v}^{\oplus,2}$ are just scaled versions of each other, hence the cos of the angle between them is equal to 1, and thus prospective index is also equal to 1 (in the limit of $\kappa \to 0$).

2.3.2 Weights in PCN are updated towards a configuration with prospective index of one

As seen in Extended Data Fig. 1d, the prospective index for layers $l > 2$ is very close to one. To provide an intuition for why this is the case, in this section we demonstrate how PCNs would need to be modified to have prospective index equal to 1. We will refer to such modified model as target-PCN, and calculate its prospective index.

As in the previous section, we assume that the model does not make a perfect prediction with the current weights, so that the error in the prediction drives the learning. We start with recapping what happens in sequence in one iteration of the standard PCN.

1. Start from relaxation with only input neurons clamped to input pattern ($\ominus$) and with current weight $\boldsymbol{W}$, the hidden neuron activity settles to: $\boldsymbol{x}_{\boldsymbol{W}}^{\ominus,l}$
2. Both input and output neurons are clamped to the input and target pattern respectively ($\oplus$) and then the hidden neuron activity is relaxed to: $\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}$
3. Weights $\boldsymbol{W}$ are updated for one step to $\boldsymbol{W}'$ to decrease the energy, while hidden neuron activity stays still from the last step: $\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}$
4. Output neurons are freed but the input neuron is still clamped to the input pattern and then the hidden neuron activity is relaxed to: $\boldsymbol{x}_{\boldsymbol{W}'}^{\ominus,l}$

In the above step 3, weights are updated for one step from $\boldsymbol{W}$ to $\boldsymbol{W}'$. However, one can investigate the case of updating weights $\boldsymbol{W}$ for many steps until convergence $\boldsymbol{W}^*$ in the above step 3. This will result in weights $\boldsymbol{W}^*$ that represents: "the target towards which the weights $\boldsymbol{W}$ are updated". Thus, we call this variant "target-PCN" and it is summarized in Algorithm 8. Specifically, the procedure of target-PCN is to replace the above steps 3 and 4 of standard PCN with:

3. Weights are updated for many steps from $\boldsymbol{W}$ to $\boldsymbol{W}^*$ to decrease the energy till convergence, while hidden neuron activity stays still from the last step: $\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}$;
4. Output neurons are freed but the input neuron is still clamped to the input pattern and then the hidden neuron activity is relaxed to: $\boldsymbol{x}_{\boldsymbol{W}^*}^{\ominus,l}$;

Algorithm 8: Learn with target-PCN

Input: input pattern $\boldsymbol{s}^{\text{in}}$; target pattern $\boldsymbol{s}^{\text{target}}$; synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$
Output: updated synaptic weights $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$

$\boldsymbol{x}^1 = \boldsymbol{s}^{\text{in}}$; // Clamp input neurons to input pattern
$\boldsymbol{x}^{L+1} = \boldsymbol{s}^{\text{target}}$; // Clamp output neurons to target pattern
for $t = 0;\ t < \mathscr{T};\ t = t+1$ **do** // Relaxation
  for $l = 1;\ l < L+1;\ l = l+1$ **do**
    $\hat{\boldsymbol{x}}^{l+1} = \boldsymbol{w}^l f\left(\boldsymbol{x}^l\right)$;
    $\boldsymbol{\varepsilon}^{l+1} = \boldsymbol{x}^{l+1} - \hat{\boldsymbol{x}}^{l+1}$;
  end
  for $l = 2;\ l < L+1;\ l = l+1$ **do**
    $\Delta \boldsymbol{x}^l = \gamma\left(-\boldsymbol{\varepsilon}^l + f'\left(\boldsymbol{x}^l\right) \circ \left(\left(\boldsymbol{w}^l\right)^T \boldsymbol{\varepsilon}^{l+1}\right)\right)$;
    $\boldsymbol{x}^l = \boldsymbol{x}^l + \Delta \boldsymbol{x}^l$;
  end
end
while $\left\{\boldsymbol{w}^1, \boldsymbol{w}^2, \cdots, \boldsymbol{w}^L\right\}$ *not converged* **do** // Update weights till convergence
  for $l = 1;\ l < L+1;\ l = l+1$ **do**
    $\hat{\boldsymbol{x}}^{l+1} = \boldsymbol{w}^l f\left(\boldsymbol{x}^l\right)$;
    $\boldsymbol{\varepsilon}^{l+1} = \boldsymbol{x}^{l+1} - \hat{\boldsymbol{x}}^{l+1}$;
  end
  for $l = 1;\ l < L+1;\ l = l+1$ **do** // Update weights
    $\Delta \boldsymbol{w}^l = \alpha \boldsymbol{\varepsilon}^{l+1}\left(f\left(\boldsymbol{x}^l\right)\right)^T$;
    $\boldsymbol{w}^l = \boldsymbol{w}^l + \Delta \boldsymbol{w}^l$;
  end
end

In the following, we demonstrate prospective index of target-PCN is one for all layers. First, we should notice that the minimum of energy E of PCN is zero, since the energy function is a sum of quadratic terms, i.e., Eq. (6). Then, we should notice that such energy E of PCN can be optimized to its minimum of zero by optimizing only $\boldsymbol{W}$. Particularly, the local energy term of layer l is:

$$\begin{aligned}\frac{1}{2}\left(\boldsymbol{\varepsilon}^l\right)^T \boldsymbol{\varepsilon}^l &= \frac{1}{2}\left(\boldsymbol{x}^l - \hat{\boldsymbol{x}}^l\right)^T \left(\boldsymbol{x}^l - \hat{\boldsymbol{x}}^l\right) \\ &= \frac{1}{2}\left(\boldsymbol{x}^l - \boldsymbol{w}^{l-1} f\left(\boldsymbol{x}^{l-1}\right)\right)^T \left(\boldsymbol{x}^l - \boldsymbol{w}^{l-1} f\left(\boldsymbol{x}^{l-1}\right)\right)\end{aligned} \tag{27}$$

In the above Eq., $\boldsymbol{x}^l - \boldsymbol{w}^{l-1} f\left(\boldsymbol{x}^{l-1}\right)$ can be optimized to produce a zero vector by optimizing only $\boldsymbol{w}^{l-1}$, as long as $f\left(\boldsymbol{x}^{l-1}\right)$ is not a zero vector. Specifically, let us denote all the non-zero entries in $f\left(\boldsymbol{x}^{l-1}\right)$ by $\left\{f\left(x_i^{l-1}\right)\right\}_{i \in I}$, where I is the set of indices i so that $f\left(x_i^{l-1}\right)$ is non-zero. Since $f\left(\boldsymbol{x}^{l-1}\right)$ is not a zero vector, $I \neq \emptyset$. To demonstrate that there exists a solution for $\left\{w_{j,i}^{l-1}\right\}_{i \in I}$ so that $x_j^l = \sum_{i \in I} w_{j,i}^{l-1} f\left(x_i^{l-1}\right)$, we construct an example of such solution. Such sample solution is to pick one index g from I, then have $w_{j,g}^{l-1} = \frac{x_j^l}{f\left(x_i^{l-1}\right)}$ and $\left\{w_{j,i}^{l-1} = 0 : i \in I, i \notin \{g\}\right\}$. Thus, as long as $f\left(\boldsymbol{x}^{l-1}\right)$ is not a zero vector ($I \neq \emptyset$), there exists a solution of $\boldsymbol{w}^{l-1}$ that makes $\boldsymbol{x}^l - \boldsymbol{w}^{l-1} f\left(\boldsymbol{x}^{l-1}\right)$ a zero vector.

Thus, in step 3 of the target-PCN, the energy of the network is at its minimum of zero. This further implies that in the step 4 of the target-PCN, the neural activity does not move, i.e.,

$$\boldsymbol{x}_{\boldsymbol{W}^*}^{\ominus,l} = \boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l} \tag{28}$$

According to the definition of prospective index in Extended Data Fig. 1a-b, the prospective index of this target-PCN ($\phi^{*,l}$) is:

$$\begin{aligned}
\phi^{*,l} &= \frac{\boldsymbol{v}^{\oplus,l} \cdot \boldsymbol{v}^{*,l}}{\left(||\boldsymbol{v}^{\oplus,l}|| + \kappa\right)\left(||\boldsymbol{v}^{*,l}|| + \kappa\right)} \\
&\approx cos\left(\boldsymbol{v}^{\oplus,l}, \boldsymbol{v}^{*,l}\right) \\
&= cos\left(\overrightarrow{\boldsymbol{x}_{\boldsymbol{W}}^{\ominus,l}\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}}, \overrightarrow{\boldsymbol{x}_{\boldsymbol{W}}^{\ominus,l}\boldsymbol{x}_{\boldsymbol{W}^*}^{\ominus,l}}\right) \\
&= cos\left(\overrightarrow{\boldsymbol{x}_{\boldsymbol{W}}^{\ominus,l}\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}}, \overrightarrow{\boldsymbol{x}_{\boldsymbol{W}}^{\ominus,l}\boldsymbol{x}_{\boldsymbol{W}}^{\oplus,l}}\right) \text{ according to Eq. (28)} \\
&= 1
\end{aligned} \tag{29}$$

This theoretical result is further confirmed by empirical observation in Extended Data Fig. 1d. Since the standard PCN modifies the weights in a similar direction as target-PCN, it is likely to have a similar prospective index.

2.4 Relationships of predictive coding networks to target propagation (Extended Data Fig. 2)

In Extended Data Fig. 2, we illustrate that prospective configuration, particularly, *predictive coding network*[30,43,55] (PCN), has close a relationship to a separate and influential model of credit assignment – target propagation[60]. Here we formally prove these observations. Particularly, we show that

- In an output-constrained PCN, neural activity after relaxation converges to the local target;
- In an input-output-constrained PCN, neural activity after relaxation converges to the weighted sum of the predicting activity and the local target.

In the above, predicting activity refer to the neural activity when the model is making prediction, and they are the same for both backpropagation and PCN as they compute the same neural activity when making a prediction, and local target are described in the following subsection.

2.4.1 Target-propagation

We first briefly review target propagation[60]. The key insight behind target propagation is that rather than updating weights based on a gradient of a loss function, one can instead attempt to explicitly compute what are the optimal activity for the neurons so that they can produce the desired target pattern, and then update the weights so as to nudge the current neural activity towards the optimal activity directly. We call these optimal activity *local target* since if the neurons takes this activity, the network would produce the desired target pattern. Importantly, we can directly compute the local target in terms of the *inverses* of the weights and activation functions. Namely, suppose that we have a three-layer network with activation functions $f()$, weight matrices $\boldsymbol{w}^1, \boldsymbol{w}^2, \boldsymbol{w}^3$ and an input pattern $\boldsymbol{s}^{\text{in}}$. The output of this network is $\boldsymbol{x}^4 = \boldsymbol{w}^3 f\left(\boldsymbol{w}^2 f\left(\boldsymbol{w}^1 f\left(\boldsymbol{s}^{\text{in}}\right)\right)\right)$. Suppose instead that we do not want the network to output $\boldsymbol{x}^4$ for a given $\boldsymbol{s}^{\text{in}}$ but rather a given target pattern $\boldsymbol{s}^{\text{target}}$. Then, the activity at the first layer

$\tilde{\boldsymbol{x}}^1$ that would produce this desired activity can be exactly computed by inverting[1] the network $\tilde{\boldsymbol{x}}^1 = f^{-1}\left(\left(\boldsymbol{w}^1\right)^{-1} f^{-1}\left(\left(\boldsymbol{w}^2\right)^{-1} f^{-1}\left(\left(\boldsymbol{w}^3\right)^{-1} \boldsymbol{s}^{\text{target}}\right)\right)\right)$. From this, we can define a recursion of one local target in terms of another at the layer above,

$$\begin{aligned} \tilde{\boldsymbol{x}}^l &= f^{-1}\left(\left(\boldsymbol{w}^l\right)^{-1} \tilde{\boldsymbol{x}}^{l+1}\right) \\ \tilde{\boldsymbol{x}}^{L+1} &= \boldsymbol{s}^{\text{target}} \end{aligned} \tag{30}$$

Like prospective configuration, these targets are also *prospective*, in some sense, since they represent the counterfactual what the neural activity *should have been* in order to produce the desired target pattern.

2.4.2 Output-constrained PCN

In this subsection we investigate the "output-constrained PCN": in this PCN input neurons are not clamped to the input pattern but output neurons are clamped to the target pattern. We show that in this PCN, the activity after relaxation is precisely equal to the local target. Since $\boldsymbol{x}^1$ is not constrained to the input pattern, we can look at its dynamic by setting $l = 1$ in Eq. (12). Since there is no error term or error nodes at the input layer, there is only the later term left when setting $l = 1$ in Eq. (12) (note that here we write in matrix & vector form):

$$\Delta \boldsymbol{x}^1 = \gamma f'\left(\boldsymbol{x}^1\right) \circ \left(\left(\boldsymbol{w}^1\right)^T \boldsymbol{\varepsilon}^2\right) \tag{31}$$

$$= \gamma f'\left(\boldsymbol{x}^1\right) \circ \left(\left(\boldsymbol{w}^1\right)^T \left(\boldsymbol{x}^2 - \boldsymbol{w}^1 f\left(\boldsymbol{x}^1\right)\right)\right) \tag{32}$$

Considering the above dynamic has converged, we can set $\Delta \boldsymbol{x}^1 = \boldsymbol{0}$ in the above equation and solving for $\boldsymbol{x}^1$, then we can obtain the converged value of $\boldsymbol{x}^1$:

$$\boldsymbol{x}^1 = f^{-1}\left(\left(\boldsymbol{w}^1\right)^{-1} \boldsymbol{x}^2\right) \tag{33}$$

Now we look at the dynamic of $\boldsymbol{x}^2$ by setting $l = 2$ in Eq. (12):

$$\Delta \boldsymbol{x}^2 = \gamma\left(-\boldsymbol{\varepsilon}^2 + f'\left(\boldsymbol{x}^2\right) \circ \left(\left(\boldsymbol{w}^2\right)^T \boldsymbol{\varepsilon}^3\right)\right) \tag{34}$$

$$= \gamma\left(-\left(\boldsymbol{x}^2 - \boldsymbol{w}^1 f\left(\boldsymbol{x}^1\right)\right) + f'\left(\boldsymbol{x}^2\right) \circ \left(\left(\boldsymbol{w}^2\right)^T \left(\boldsymbol{x}^3 - \boldsymbol{w}^2 f\left(\boldsymbol{x}^2\right)\right)\right)\right) \tag{35}$$

Putting the solved $\boldsymbol{x}^1$, i.e., Eq. (33), into the above Eq., we have:

$$\Delta \boldsymbol{x}^2 = \gamma f'\left(\boldsymbol{x}^2\right) \circ \left(\left(\boldsymbol{w}^2\right)^T \left(\boldsymbol{x}^3 - \boldsymbol{w}^2 f\left(\boldsymbol{x}^2\right)\right)\right) \tag{36}$$

Considering the above dynamic has converged, we can set $\Delta \boldsymbol{x}^2 = \boldsymbol{0}$ in the above equation and solving for $\boldsymbol{x}^2$, then we can obtain the converged value of $\boldsymbol{x}^2$:

$$\boldsymbol{x}^2 = f^{-1}\left(\left(\boldsymbol{w}^2\right)^{-1} \boldsymbol{x}^3\right) \tag{37}$$

[1] Note that in realistic networks the weight matrices are not all square so an exact inverse $\left(\boldsymbol{w}^l\right)^{-1}$ does not exist. Instead, we can compute approximations of the inverse using the Moore-Penrose pseudoinverse[115] $\left(\boldsymbol{w}^l\right)^{\dagger}$, which is the least squares solution to the optimization problem $\arg\min_{\boldsymbol{w}} \left\|\boldsymbol{I} - \boldsymbol{w}^l \boldsymbol{w}\right\|$.

One can now see the proof goes recursively until $l = L$ and $\boldsymbol{x}^{L+1}$ is fixed to the target pattern $\boldsymbol{s}^{\text{target}}$:

$$\begin{aligned} \boldsymbol{x}^l &= f^{-1}\left(\left(\boldsymbol{w}^l\right)^{-1}\boldsymbol{x}^{l+1}\right) \\ \boldsymbol{x}^{L+1} &= \boldsymbol{s}^{\text{target}} \end{aligned} \tag{38}$$

which is exactly the recursive formula of the local target in target propagation, i.e., Eq. (30). Thus, neural activity of output-constrained PCN after relaxation equals to the local target.

2.4.3 Input-output-constrained PCN

In this subsection we investigate the "input-output-constrained PCN": in this PCN both input and output neurons are clamped to the input and target patterns, respectively. We show that in this PCN, the activity after relaxation are the weighted sum of the predicting activity and the local target. Particularly, since in a input-output-constrained PCN, we can only solve for the equilibrium after relaxation analytically in the linear case, we prove this for a linear PCN. Nevertheless, the analysis still provides useful insights. Looking at the network dynamics at a given layer l, i.e., Eq. (12), we can write the dynamics in the linear case as,

$$\Delta\boldsymbol{x}^l = \gamma\left(-\left(\boldsymbol{x}^l - \boldsymbol{w}^{l-1}\boldsymbol{x}^{l-1}\right) + \left(\boldsymbol{w}^l\right)^T\left(\boldsymbol{x}^{l+1} - \boldsymbol{w}^l\boldsymbol{x}^l\right)\right) \tag{39}$$

If we then set $\Delta\boldsymbol{x}^l = \boldsymbol{0}$ and solve for $\boldsymbol{x}^l$, we obtain,

$$\Delta\boldsymbol{x}^l = \boldsymbol{0} \implies -\left(\boldsymbol{x}^l - \boldsymbol{w}^{l-1}\boldsymbol{x}^{l-1}\right) + \left(\boldsymbol{w}^l\right)^T\left(\boldsymbol{x}^{l+1} - \boldsymbol{w}^l\boldsymbol{x}^l\right) = \boldsymbol{0} \tag{40}$$

$$\implies -\boldsymbol{x}^l + \boldsymbol{w}^{l-1}\boldsymbol{x}^{l-1} + \left(\boldsymbol{w}^l\right)^T\boldsymbol{x}^{l+1} - \left(\boldsymbol{w}^l\right)^T\boldsymbol{w}^l\boldsymbol{x}^l = \boldsymbol{0} \tag{41}$$

$$\implies \boldsymbol{x}^l + \left(\boldsymbol{w}^l\right)^T\boldsymbol{w}^l\boldsymbol{x}^l = \boldsymbol{w}^{l-1}\boldsymbol{x}^{l-1} + \left(\boldsymbol{w}^l\right)^T\boldsymbol{x}^{l+1} \tag{42}$$

$$\implies \left(\boldsymbol{I} + \left(\boldsymbol{w}^l\right)^T\boldsymbol{w}^l\right)\boldsymbol{x}^l = \boldsymbol{w}^{l-1}\boldsymbol{x}^{l-1} + \left(\boldsymbol{w}^l\right)^T\boldsymbol{x}^{l+1} \tag{43}$$

$$\implies \boldsymbol{x}^l = \left(\boldsymbol{I} + \left(\boldsymbol{w}^l\right)^T\boldsymbol{w}^l\right)^{-1}\left(\boldsymbol{w}^{l-1}\boldsymbol{x}^{l-1} + \left(\boldsymbol{w}^l\right)^T\boldsymbol{x}^{l+1}\right) \tag{44}$$

If we assume that the norm of the weights is large compared to the identity matrix $\boldsymbol{I}$, i.e., we consider $\left(\boldsymbol{I} + \left(\boldsymbol{w}^l\right)^T\boldsymbol{w}^l\right)^{-1} \approx \left(\left(\boldsymbol{w}^l\right)^T\boldsymbol{w}^l\right)^{-1}$, the above equilibrium solution can further be approximated by:

$$\implies \boldsymbol{x}^l \approx \left(\left(\boldsymbol{w}^l\right)^T\boldsymbol{w}^l\right)^{-1}\left(\boldsymbol{w}^{l-1}\boldsymbol{x}^{l-1} + \left(\boldsymbol{w}^l\right)^T\boldsymbol{x}^{l+1}\right) \tag{45}$$

$$\implies \boldsymbol{x}^l \approx \left(\left(\boldsymbol{w}^l\right)^T\boldsymbol{w}^l\right)^{-1}\underbrace{\boldsymbol{w}^{l-1}\boldsymbol{x}^{l-1}}_{\substack{\text{predicting activity} \\ \text{for backpropagation and PCN}}} + \underbrace{\left(\boldsymbol{w}^l\right)^{-1}\boldsymbol{x}^{l+1}}_{\substack{\text{local target} \\ \text{from target propagation}}} \tag{46}$$

where the equilibrium solution is simply the weighted sum of the predicting activity and the local target.

These theoretical results let us view prospective configuration from another angle. Namely, that prospective configuration tends to move the activity from the predicting activity towards the local target

that would be computed by target propagation. These local target are intrinsically prospective in the sense that they are the neural activity which, if they were produced, they would minimize the loss function (i.e., produce the desired target pattern). In other words, they are in some sense the activity which the weight updates of a network are trying to achieve. By moving towards these local target directly in activity-space, as opposed to only in weight-space like backpropagation, the prospective configuration can often propose faster and more efficient updates than can be achieved by backpropagation.