Neural optimal feedback control with local learning rules

Friedrich

Optimal Feedback Control (OFC)

- Optimal Feedback Control (OFC) is a framework from control theory that describes how to compute actions (or controls) to achieve a goal in the most efficient way, especially in systems with uncertainty, like noise or delayed feedback.
    - OFC continuously adjusts actions based on real-time sensory feedback, using an internal model to predict future states and minimize a cost (like error or effort).

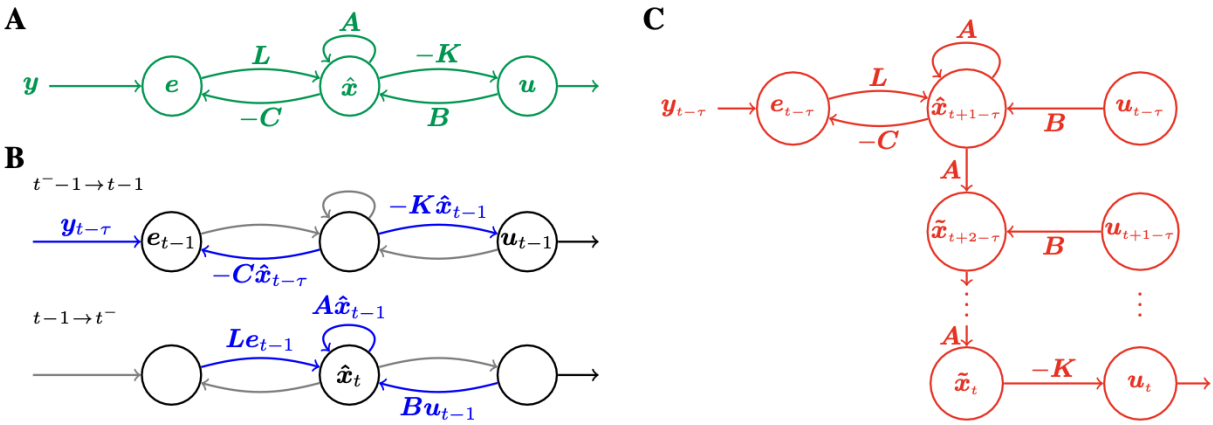

Figure S1: **Schematic of our proposed neural networks for OFC and alternative artificial neural network. (A)** Schematic of our proposed neural network (Bio-OFC). Nodes are annotated with the quantity they represent in their firing rates; edges are annotated with synaptic weights. **(B)** Input currents when updating $\boldsymbol{e}_{t-1}$, $\boldsymbol{u}_{t-1}$ (top), and $\hat{\boldsymbol{x}}_t$ (bottom). $\hat{\boldsymbol{x}}_t$ is updated directly using the delayed measurement $\boldsymbol{y}_{t-\tau}$, even for delays $\tau > 1$. **(C)** The alternative artificial neural network (ANN) that updates the past estimate $\hat{\boldsymbol{x}}_{t+1-\tau}$, and predicts forward in time to estimate $\tilde{\boldsymbol{x}}_{t+2-\tau}, \ldots, \tilde{\boldsymbol{x}}_t$, would require biologically implausible weight copies of $\boldsymbol{A}$ and $\boldsymbol{B}$, as well as a memory of past controls $\boldsymbol{u}_{t-\tau}, \ldots, \boldsymbol{u}_{t-1}$.

-

Kalman Filtering and Linear-Quadratic Regulator (LQR)

- Classical approaches like Kalman filtering and linear-quadratic regulators (LQR) work well in theory, but they're not biologically plausible.
- The **Kalman Filter (for Estimation)** is an algorithm used to **estimate** the hidden state of a system (like position or velocity) from noisy observations.
    - It combines predictions from a model with actual noisy observations to get the best estimate of the current state.
    - Example: You're tracking a moving object (like your hand). You don’t see it clearly due to noisy visual input, but you can predict where it’s going based on

how it was moving. The Kalman filter fuses that prediction and the noisy observation to guess where your hand really is.

- The **Linear-Quadratic Regulator (LQR) (for Control)** is a method to compute the optimal action (**control**) to apply to a system to minimize a cost, assuming the system's dynamics and noise are known and linear.
    - Given a current estimate of the state, it outputs a control action that minimizes a long-term cost function (usually quadratic—e.g., penalties on error and effort).
    - Example: It helps you smoothly move your hand to a target while using the least amount of energy and keeping the motion accurate.
- Why these aren't biologically plausible:
    - They require exact knowledge of system dynamics and noise statistics.
    - They often involve matrix operations, like inversion and solving Riccati equations—hard to do with real neurons.
    - They assume instant access to clean data (**no delays**) and global updates across the system.
    - Learning in these frameworks isn't local (i.e., neurons can't just use their own input/output to update weights).
- So, while Kalman filters and LQRs are mathematically elegant, the brain probably doesn't implement them directly, which is why papers like this one try to build **neural approximations that use biologically plausible rules**.

Core Idea of Bio-OFC

- The authors aim to model how the brain might solve control problems like moving your hand to a target despite sensory noise and delay, without needing to know the exact equations of motion or noise levels. Classical approaches like Kalman filtering and linear-quadratic regulators (LQR) work well in theory, but they're not biologically plausible.
- So they build a network that:
    - Uses adaptive Kalman filtering for estimating the system state from delayed/noisy observations.
    - Uses policy gradient (a reinforcement learning method) for learning a control policy, avoiding backpropagation and non-local updates.
    - Learns both the system dynamics (A, B, C) and Kalman gain (L) using online local learning rules, meaning updates are only based on information locally available at the synapse.

1. **Bio-OFC Architecture**:
    - It contains neurons representing:
        - **State estimates** ($\hat{x}_t$),
        - **Prediction errors** (e_t),
        - **Control actions** (u_t).
    - **Kalman filtering is modified** to work with delayed observations ($y_{t-\tau}$).
2. **Learning Rules**:
    - **Local plasticity rules** update system parameters using Hebbian-like updates.
    - Control weights are updated using **policy gradients with eligibility traces**, simulating dopamine-like global signals.
3. **No Prior Knowledge Assumed**:
    - Doesn't assume knowledge of noise covariances or dynamics.
    - Works **online** with **single-phase learning** (no alternating "learning" and "execution" phases).
4. **Biological Plausibility**:
    - **No weight transport** or centralized computation.
    - Supports **delayed feedback**, something prior models ignored.
5. **Experiments**:
    - **Double integrator system** (a basic control problem),
    - **Human hand-reaching task** (comparison with motor adaptation experiments),
    - **2D winged flight** (tests performance in nonlinear and delayed scenarios).

Table 1: **Limitations of previously proposed neural implementations of OFC.** Presence or absence of different properties in previously proposed neural models, and their comparison to Bio-OFC. Guide to symbols: ✓: true, ✗: false, ✓✗: partially true, N/A: not applicable.

	[4]	[6]	[5]	[7]	[8]	Bio-OFC
delayed sensory feedback	✗	✗	✗	✗	✗	✓
control included	✗	✓	✗	✗	✗	✓
noise covariance agnostic	✗	✓	✗	✗	✗	✓
online system identification	✗	✓	✗	✓	✓✗	✓
local learning rules	N/A	✓	N/A	✗	✓	✓
tractable latent size	✗	✓	✗	✓	✓	✓
absence of inner loop	✓	✗	✓	✓	✗	✓
single phase learning/execution	N/A	✗	N/A	✓	✓	✓

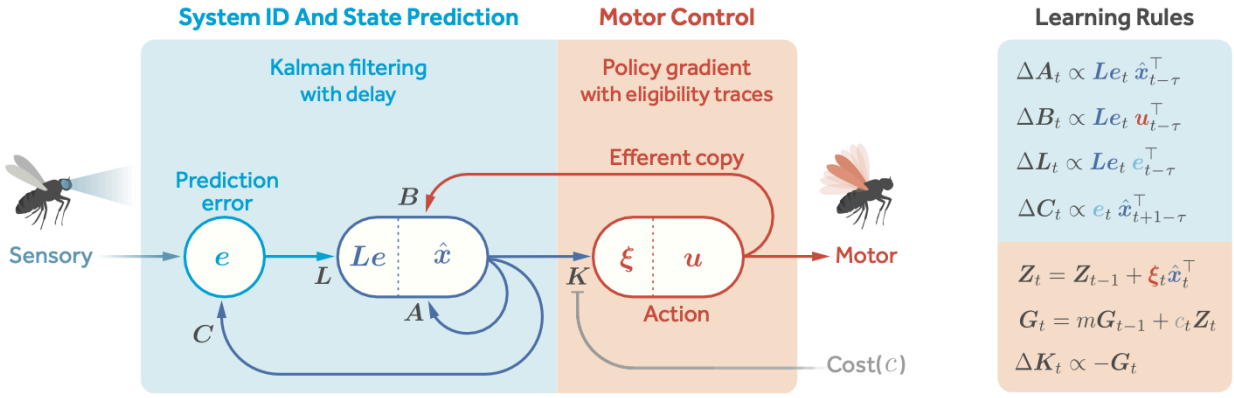

Figure 1: **The circuit and learning rules of the Bio-OFC algorithm.** Our circuit is comprised of two main parts. First (in blue), the circuit performs Kalman filtering. Then (in red), the circuit performs control using policy gradients with eligibility traces. Triangular arrowheads denote synaptic connections and the flat arrowhead denotes the modulatory effect of the cost signal.

Problem Formulation

2.1 Problem formulation

We model the environment as a linear dynamical system driven by control input and perturbed by Gaussian noise. The true state of the system $\boldsymbol{x}$ is hidden and all the animal has access to are the observations $\boldsymbol{y}$ that are assumed to be linear functions of the state corrupted by Gaussian noise.

$$\text{dynamics:} \qquad \boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{v}_t \tag{1}$$

$$\text{observation:} \qquad \boldsymbol{y}_t = \boldsymbol{C}\boldsymbol{x}_t + \boldsymbol{w}_t \tag{2}$$

Here $\boldsymbol{v}_t \sim \mathcal{N}(0; \boldsymbol{V})$ and $\boldsymbol{w}_t \sim \mathcal{N}(0, \boldsymbol{W})$ are independent Gaussian random variables and the initial state has a Gaussian prior distribution $\boldsymbol{x}_0 \sim \mathcal{N}(\hat{\boldsymbol{x}}_0, \boldsymbol{\Sigma}_0)$.

The goal is to estimate the latent state $\hat{\boldsymbol{x}}$ in order to design a control $\boldsymbol{u}$ that minimizes expected cost

$$\text{expected cost:} \qquad J = \mathbb{E}\left[\sum_{t=0}^{T} c(\boldsymbol{x}_t, \boldsymbol{u}_t)\right] \tag{3}$$

$$\text{control:} \qquad \boldsymbol{u}_t = k(\hat{\boldsymbol{x}}_t) = \arg\min J \tag{4}$$

where $c(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is the instantaneous cost associated with state $\boldsymbol{x}_t$ and action $\boldsymbol{u}_t$. As the environment dynamics is not known to the animal a priori, the parameters $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ must be learned online.

1. Dynamics of the system (Eq. 1):

$$x_{t+1} = Ax_t + Bu_t + v_t$$

- x_t: the **hidden state** at time t (e.g., position and velocity of a limb).
- u_t: the **control input** at time t (e.g., muscle activation).
- A: state transition matrix (how state evolves on its own).
- B: control matrix (how control input affects the state).
- v_t: **process noise**, drawn from a Gaussian distribution with covariance V.
  - This equation describes how the system evolves over time under control and noise.

2. Observation model (Eq. 2):

$$y_t = Cx_t + w_t$$

- y_t: **observed signal**, like sensory feedback.
- C: observation matrix (how the hidden state is projected into observations).
- w_t: **observation noise**, also Gaussian with covariance W.
  - You don't see the true state x_t; instead, you observe a noisy version of it, y_t.

3. Objective:

Estimate the **hidden state** $\hat{x}_t$ (using something like Kalman filtering), and use it to generate a control input u_t that minimizes the **expected cumulative cost**:

$$J = \mathbb{E}\left[\sum_{t=0}^{T} c(x_t, u_t)\right]$$

- $c(x_t, u_t)$: cost function — penalizes being far from the goal, or using too much control (like energy).
- Goal: choose $u_t = k(\hat{x}_t)$ to minimize this cost over time.

- Challenge: You don't know the system parameters A, B, and C. The system must learn them online from data – just like a brain learning how to move in a new environment.
- Recap

- A: State transition matrix — tells you how the current state x_t evolves into the next state x_{t+1} **without any control input**.
  - > Example: if you're coasting on a bike, A predicts where you'll go just based on your momentum.
- B: Control matrix — tells you how your **actions** u_t (like pushing pedals) influence the next state.
  - > Larger B means more effect from your inputs.
- C: Observation matrix — maps the true state x_t to the **observable signals** y_t.
  - > Think of it like a camera or a sensor that only shows a noisy version of what's really going on.

- **Online Learning**
  - "Online" learning means the system updates its knowledge step-by-step as data comes in — it doesn't wait for all the data to be collected.
  - The system does not know how the world works ahead of time (i.e., it doesn't know A, B, C), so it has to infer them while interacting with the environment.
- In classical control, you assume A,B,C are known — you can compute everything ahead of time.
- But in the brain (or in real adaptive systems), you have to figure them out on the fly based on:
  - What you did (u_t)
  - What you saw (y_t)
  - And your estimate of what the system's state was ($\hat{x}_t$)
- This is also called system identification — learning the internal model of how actions and the environment evolve.
- In this paper, the authors develop local learning rules (i.e. biologically plausible updates) that allow a neural network to gradually estimate these matrices just using its **local activity** — which is pretty cool.

- **Local Activity**
  - By local activity, we mean information that's available at a single neuron and its connections — specifically:
    - The neuron's own activity (e.g. firing rate),
    - Activity of its presynaptic (input) and postsynaptic (output) partners,
    - Possibly a global modulatory signal (like dopamine).
  - In short: no need to access other parts of the network or perform matrix-wide computations. That's what makes a learning rule biologically plausible.

Kalman Estimation and Control

2.2 Kalman estimation and control

The Kalman filter [16, 2] is an estimator of the latent state $\hat{\boldsymbol{x}}_t$ (and its variance) via a weighted summation of the current observation and the prediction of the internal model based on prior measurements. However, in biologically realistic situations the sensory feedback is always delayed. This means that the control signal $\boldsymbol{u}_t$ has to be issued, and thus the state $\boldsymbol{x}_t$ estimated, before $\boldsymbol{y}_t$ has been observed. The appropriately modified Kalman filter computes the posterior probability distribution of $\boldsymbol{x}_t$ given observations $\boldsymbol{y}_{t-\tau}, ..., \boldsymbol{y}_0$ where $\tau \geq 1$ is the delay. We start here with the case of $\tau = 1$ for which the recursive updates of the mean $\hat{\boldsymbol{x}}_t$ and the variance $\boldsymbol{\Sigma}_t$ are

$$\hat{\boldsymbol{x}}_{t+1} = \boldsymbol{A}\hat{\boldsymbol{x}}_t + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{L}_t(\boldsymbol{y}_t - \boldsymbol{C}\hat{\boldsymbol{x}}_t) \tag{5}$$

$$\boldsymbol{\Sigma}_{t+1} = (\boldsymbol{A} - \boldsymbol{L}_t\boldsymbol{C})\boldsymbol{\Sigma}_t\boldsymbol{A}^\top + \boldsymbol{V} \tag{6}$$

where $\boldsymbol{L}_t$ is known as the Kalman gain matrix which optimally combines the noisy observations $\boldsymbol{y}_t$ with the internal model and is given by

$$\boldsymbol{L}_t = \boldsymbol{A}\boldsymbol{\Sigma}_t\boldsymbol{C}^\top(\boldsymbol{C}\boldsymbol{\Sigma}_t\boldsymbol{C}^\top + \boldsymbol{W})^{-1}. \tag{7}$$

The Kalman filter is optimal in the sense that it minimizes the mean-squared error $\mathbb{E}[\boldsymbol{e}_t\top\boldsymbol{e}_t]$ with prediction error (innovation) $\boldsymbol{e}_t = \boldsymbol{y}_t - \boldsymbol{C}\hat{\boldsymbol{x}}_t$.

The output feedback law, also known as policy, (4) simplifies if the cost J is quadratic in $\boldsymbol{u}_t$ and $\boldsymbol{x}_t$:

$$\boldsymbol{u}_t = -\boldsymbol{K}\hat{\boldsymbol{x}}_t \tag{8}$$

and is known as linear-quadratic regulator (LQR). The control gain, $\boldsymbol{K}$, is found by solving a matrix Riccati equation (cf. Supplementary Material). Linear policies have been successfully applied to a variety of control tasks [17].

- **Estimate Part**

  (5) Update of the state estimate (mean):

  $$\hat{x}_{t+1} = A\hat{x}_t + Bu_t + L_t(y_t - C\hat{x}_t)$$

  Let's unpack this:

  - $A\hat{x}_t + Bu_t$: your **prediction** for what the next state should be.
  - $y_t - C\hat{x}_t$: the **prediction error** (aka "innovation") — how much your prediction differs from what was observed.
  - L_t: the **Kalman gain** — it decides how much to trust the observation vs. your internal model.

  (6) Update of the estimate variance (uncertainty):

  $$\Sigma_{t+1} = (A - L_tC)\Sigma_tA^\top + V$$

  This tracks your **uncertainty** about the state. As you get new observations, this should ideally go down.

(7) The Kalman gain formula:

$$L_t = A\Sigma_t C^\top (C\Sigma_t C^\top + W)^{-1}$$

This formula decides how to **weight your prediction error** when updating your belief. It adapts based on:

- Your confidence in your current state estimate (Σ_t),
- And how noisy the observations are (W).

- Kalman filtering is optimal in the sense that it minimizes the mean squared prediction error, $e_t = y_t - C\hat{x}_t$

- **Control Part**

  Linear-Quadratic Regulator (LQR) is a classic method in control theory for computing the **optimal way to act** (i.e., choose control inputs u_t) in a linear system so that you minimize a given **cost function** over time.

  It's called:

  - **Linear**: because the system dynamics are linear (as in $x_{t+1} = Ax_t + Bu_t + \text{noise}$),
  - **Quadratic**: because the cost function is quadratic in both the state and control.

  Quadratic cost functions are used because they:

  1. **Penalize deviation** from the desired state (like being far from a goal),
  2. **Penalize excessive control effort** (e.g., you don't want to use too much energy).

  A typical form of the cost is:

$$J = \sum_{t=0}^{T} \left(x_t^\top Q x_t + u_t^\top R u_t \right)$$

  - $x_t^\top Q x_t$: penalizes how far the state is from zero (or a goal), scaled by matrix Q,
  - $u_t^\top R u_t$: penalizes large control inputs, scaled by R.

- Quadratic terms are nice because they're:
  - Smooth and convex → easy to optimize,
  - Closed-form solutions exist (e.g., for computing K).

  (8) Control policy:

$$u_t = -K\hat{x}_t$$

  - K is the **control gain** — tells you how strongly to respond to the estimated state.
  - This is known as a **Linear Quadratic Regulator (LQR)**.

- More details:

- K is the **control gain matrix**.
- It tells you **how strongly to react to your current estimated state**.
- It is computed based on the system dynamics and the cost matrices Q, R by solving the **Riccati equation** (a specific matrix equation).

In practice:

- If K is large, the controller reacts **aggressively** to deviations (big corrections),
- If K is small, it's more **cautious** (gentle corrections).

A Linear-quadratic regulator (LQR)

The optimal control problem is to determine an output feedback law that minimizes the expected value of a cost criterion. If the cost J is quadratic, the optimal output feedback is a linear control law known as linear-quadratic regulator (LQR).

$$J = \boldsymbol{x}_T^\top \boldsymbol{Q} \boldsymbol{x}_T + \sum_{t=0}^{T-1} \left(\boldsymbol{x}_t^\top \boldsymbol{Q} \boldsymbol{x}_t + \boldsymbol{u}_t^\top \boldsymbol{R} \boldsymbol{u}_t \right) \tag{S1}$$

$$\boldsymbol{u}_t = -\boldsymbol{K}_t \boldsymbol{x}_t \quad \text{with control gain} \quad \boldsymbol{K}_t = (\boldsymbol{B}^\top \boldsymbol{P}_{t+1} \boldsymbol{B} + \boldsymbol{R})^{-1} \boldsymbol{B}^\top \boldsymbol{P}_{t+1} \boldsymbol{A} \tag{S2}$$

where $\boldsymbol{P}_t$ is determined by the dynamic Riccati equation that runs backwards in time

$$\boldsymbol{P}_{t-1} = \boldsymbol{A}^\top \boldsymbol{P}_t \boldsymbol{A} - (\boldsymbol{A}^\top \boldsymbol{P}_t \boldsymbol{B}) \left(\boldsymbol{R} + \boldsymbol{B}^\top \boldsymbol{P}_t \boldsymbol{B} \right)^{-1} (\boldsymbol{B}^\top \boldsymbol{P}_t \boldsymbol{A}) + \boldsymbol{Q} \tag{S3}$$

from terminal condition $\boldsymbol{P}_T = \boldsymbol{Q}$. Linear-quadratic-Gaussian (LQG) control uses the Kalman estimate $\hat{\boldsymbol{x}}$ in the controller, $\boldsymbol{u}_t = -\boldsymbol{K}_t \hat{\boldsymbol{x}}_t$.

- What does LQR give you?
  - A **feedback controller**: it maps your state estimate $\hat{x}_t$ directly to the control input u_t,
  - It's **optimal**: minimizes the total cost over time,
  - And it's **efficient**: once K is computed, control is just a simple matrix multiplication at every step.

- **Linear-Quadratic-Gaussian (LQG) Control**

  This section connects **estimation (Kalman filter)** with **action (LQR)**:

  1. Kalman filter gives you the best estimate of what's happening.
  2. LQR tells you the best way to act on it.

  Together, this is called **Linear-Quadratic-Gaussian (LQG) control** — powerful, but not biologically realistic unless you find ways to approximate it (which is what this paper is trying to do).

Neural Network Representation for Optimal Feedback Control

Inference:

3 Neural network representation for optimal feedback control

3.1 Inference

For now, let us assume that the system dynamics, the Kalman gain and the control gain $\boldsymbol{A}, \boldsymbol{B}, -\boldsymbol{C}$, $\boldsymbol{L}$, and $\boldsymbol{K}$ are constant and known. Then, the latent state $\hat{\boldsymbol{x}}$ can be obtained by the Kalman estimator Eq. (5). This algorithm naturally maps onto the network in Fig. 1 and Supplementary Fig. S1A with neural populations representing $\hat{\boldsymbol{x}}$, $\boldsymbol{e} := \boldsymbol{y} - \boldsymbol{C}\hat{\boldsymbol{x}}$, and $\boldsymbol{u}$ that are connected by synapses whose weights represent the elements of the matrices $\boldsymbol{A}, \boldsymbol{B}, -\boldsymbol{C}$ and $\boldsymbol{L}$. The computation of the control variable $\boldsymbol{u}$ according to Eq. (S2) can be implemented by synapses whose weights represent the elements of the matrix $-\boldsymbol{K}$.

If the sensory stimulus delay is $\tau = 1$, our network implements the Kalman prediction reviewed in the previous section. In case $\tau > 1$, the latent state must be recomputed throughout the delay period which requires a biologically implausible circuit (cf. Supplementary Fig. S1C). To overcome this, we adapt an alternative solution from online control [18, 19]. Specifically, we combine delayed measurements with the similarly delayed latent state estimation. Such inference can be performed by the network of the same architecture and adjusting the synaptic delay associated with matrix $\boldsymbol{C}$ to match with the sensory delay, in accordance with the following expression:

$$\hat{\boldsymbol{x}}_{t+1} = \boldsymbol{A}\hat{\boldsymbol{x}}_t + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{L}\underbrace{\left(\boldsymbol{y}_{t+1-\tau} - \boldsymbol{C}\hat{\boldsymbol{x}}_{t+1-\tau}\right)}_{\boldsymbol{e}_t} \tag{9}$$

As we show in Results (Fig. 3) this reduces predictive performance only modestly compared to the biologically unrealistic scheme. More details on the inference process and the temporal order in which our recurrent network performs the above steps is shown in Supplementary Fig. S1B.

-
- If $\tau = 1$,
  - use the equation from the previous section
- If $\tau > 1$,

When the sensory observation is **delayed by** $\tau > 1$, it's biologically unrealistic to **recompute all intermediate predictions** — that would require copying weights and holding long memory (not plausible for the brain).

So the authors propose a **workaround**:

- Instead of updating $\hat{x}_{t+1}$ with the current y_t,
- They use the **delayed** observation $y_{t+1-\tau}$ and the **equally delayed prediction** $C\hat{x}_{t+1-\tau}$.

This gives the modified inference equation (Eq. 9):

$$\hat{x}_{t+1} = A\hat{x}_t + Bu_t + L\left(y_{t+1-\tau} - C\hat{x}_{t+1-\tau}\right)$$

- The expression in parentheses is still the **prediction error** e_t, just shifted to account for delay.

- This approach:
  - Preserves biological plausibility — doesn't require unrealistic memory or backtracking.
  - Still works well, even with delay (as shown in their results).
  - Can be implemented using the same neural architecture, just by adjusting synaptic delay timing.
- So in short: this section shows how to make the Kalman estimation step biologically realistic, even when the brain gets delayed feedback, which is almost always the case (e.g., it takes time for your eyes or skin to send signals).

Learning: System Identification and Kalman Gain

3.2 Learning

3.2.1 System identification and Kalman gain

Next, we turn our attention to the system identification/learning problem, which was not addressed by some of the previous proposals, cf. Table 1. Given a sequence of observations $\{\boldsymbol{y}_0, \cdots, \boldsymbol{y}_T\}$, we use a least squares approach [20] to find the parameters that minimize the mean-square prediction error $\frac{1}{T}\sum_{t=0}^{T} \boldsymbol{e}_t^\top \boldsymbol{e}_t$. We perform this optimization in an online manner, that is at each time-step $t+1$, after making the delayed observation $\boldsymbol{y}_{t+1-\tau}$, we update the parameter estimates $\hat{\boldsymbol{A}}, \hat{\boldsymbol{B}}, \hat{\boldsymbol{C}}$ using steps that minimize $\boldsymbol{e}_t^\top \boldsymbol{e}_t$, assuming that the state estimate and actions corresponding to prior observations (e.g. $\hat{\boldsymbol{x}}_{t-\tau}$, $\boldsymbol{u}_{t-\tau}$ etc.) are fixed. To obtain $\boldsymbol{L}$ we would like to avoid solving the Riccati equation (6) as it requires matrix operations difficult to implement in biology. So, we use the same optimization procedure to update the Kalman gain $\boldsymbol{L}$. Using Eq. (9) and explicitly writing out the matrix/vector indices as superscripts yields the following stochastic gradient with respect to $\boldsymbol{A}$:

$$
\begin{aligned}
-\frac{\partial}{\partial A^{ij}} \frac{1}{2} \sum_k \left(e_t^k\right)^2 &= -\sum_k e_t^k \frac{\partial e_t^k}{\partial A^{ij}} = -\sum_k e_t^k \frac{\partial \left(y_{t+1-\tau}^k - \sum_l C^{kl} \hat{x}_{t+1-\tau}^l\right)}{\partial A^{ij}} = \\
&= \sum_{k,l} e_t^k C^{kl} \frac{\partial \left(\sum_m A^{lm} \hat{x}_{t-\tau}^m + \sum_n B^{ln} u_t^n + \sum_p L^{lp} e_t^p\right)}{\partial A^{ij}} = \\
&= \sum_{k,l,m} e_t^k C^{kl} \delta^{li} \delta^{mj} \hat{x}_{t-\tau}^m = \sum_k e_t^k C^{ki} \hat{x}_{t-\tau}^j
\end{aligned} \tag{10}
$$

Performing similar derivations for the other synaptic weights, our optimization procedure would rely on the following stochastic gradients:

$$
-\nabla_{\boldsymbol{A}} \tfrac{1}{2} \boldsymbol{e}_t^\top \boldsymbol{e}_t = \boldsymbol{C}^\top \boldsymbol{e}_t \hat{\boldsymbol{x}}_{t-\tau}^\top \qquad -\nabla_{\boldsymbol{B}} \tfrac{1}{2} \boldsymbol{e}_t^\top \boldsymbol{e}_t = \boldsymbol{C}^\top \boldsymbol{e}_t \boldsymbol{u}_{t-\tau}^\top \tag{11}
$$

$$
-\nabla_{\boldsymbol{L}} \tfrac{1}{2} \boldsymbol{e}_t^\top \boldsymbol{e}_t = \boldsymbol{C}^\top \boldsymbol{e}_t \boldsymbol{e}_{t-\tau}^\top \qquad -\nabla_{\boldsymbol{C}} \tfrac{1}{2} \boldsymbol{e}_t^\top \boldsymbol{e}_t = \boldsymbol{e}_t \hat{\boldsymbol{x}}_{t+1-\tau}^\top \tag{12}
$$

This yields a classical Hebbian rule between (a memory trace of) presynaptic activity $\hat{\boldsymbol{x}}_{t+1-\tau}$ and postsynaptic activity $\boldsymbol{e}_t$ for weights $\boldsymbol{C}$. However, it suggests non-local learning rules for $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{L}$, which runs contrary to biological requirements. We can circumvent this problem by replacing $\boldsymbol{C}^\top$ with $\boldsymbol{L}$, which corresponds to left-multiplication of the gradients with a positive definite matrix (see Supplementary Material Sec. D). This still decreases the mean-square prediction error under some mild initialization constraints on $\boldsymbol{C}$ and $\boldsymbol{L}$ and yields local plasticity rules, cf. Fig. 1,

$$
\Delta \hat{\boldsymbol{A}}_t \propto \boldsymbol{L}\boldsymbol{e}_t \, \hat{\boldsymbol{x}}_{t-\tau}^\top \tag{13}
$$

$$
\Delta \boldsymbol{L}_t \propto \boldsymbol{L}\boldsymbol{e}_t \, \boldsymbol{e}_{t-\tau}^\top \tag{14}
$$

$$
\Delta \hat{\boldsymbol{B}}_t \propto \boldsymbol{L}\boldsymbol{e}_t \, \boldsymbol{u}_{t-\tau}^\top \tag{15}
$$

$$
\Delta \hat{\boldsymbol{C}}_t \propto \boldsymbol{e}_t \, \hat{\boldsymbol{x}}_{t+1-\tau}^\top \tag{16}
$$

where the input current $\boldsymbol{L}\boldsymbol{e}_t$ is locally available at neurons representing $\hat{\boldsymbol{x}}$. The first three rules are local, but non-Hebbian, capturing correlations between presynaptic activity $\hat{\boldsymbol{x}}_{t-\tau}, \boldsymbol{u}_{t-\tau}, \boldsymbol{e}_{t-\tau}$ and postsynaptic current $\boldsymbol{L}\boldsymbol{e}_t$. Note that these updates do not require knowledge of the noise covariances $\boldsymbol{V}$ and $\boldsymbol{W}$, an advantage over previous work, cf. Table 1.

The goal is **system identification**: learning the unknown parameters of the model (A, B, C, L) **online**, i.e., while the system is running — like a brain learning its own dynamics through experience.

The authors use a **least squares approach**: they try to **minimize the squared prediction error**:

$$e_t = y_{t+1-\tau} - C\hat{x}_{t+1-\tau}$$

The idea is: the more accurate your prediction $\hat{x}_{t+1-\tau}$, the smaller e_t is.

So they minimize:

- $$\frac{1}{2}e_t^\top e_t$$

They derive **gradients** of the squared error with respect to each matrix. For example:

$$\frac{\partial}{\partial A^{ij}}\left(\frac{1}{2}e_t^\top e_t\right)$$

They work out this derivative in a fully expanded matrix form in equation (10). Without diving into the math too deeply, the final result is:

- For A:

$$\Delta\hat{A}_t \propto Le_t\hat{x}_{t-\tau}^\top$$

- For B:

$$\Delta\hat{B}_t \propto Le_t u_{t-\tau}^\top$$

- For L:

$$\Delta L_t \propto Le_t e_{t-\tau}^\top$$

- For C:

- $$\Delta\hat{C}_t \propto e_t\hat{x}_{t+1-\tau}^\top$$

📍 Biologically plausible or not?

Here's the challenge:

- The initial gradients involve $C^\top$ (transpose of the observation matrix),
- This leads to **non-local learning rules** — not biologically realistic (neurons would need to know global matrix information).

🧠 **Their trick**: replace $C^\top$ with L, which is **locally available** (as input current into neurons representing $\hat{x}$).

- This substitution is mathematically valid under certain assumptions and makes the updates **local and biologically plausible** — only dependent on the activity of nearby (pre/post-synaptic) neurons.

✅ 1) How is Bio-OFC only dependent on activity of nearby neurons, compared to the other approach?

The Bio-OFC network (green, panel A + B) is biologically plausible because:

🔁 **All updates rely on local interactions:**

- Neurons represent variables like $\hat{x}_t$, $e_t = y - C\hat{x}_t$, and $u_t = -K\hat{x}_t$.
- Synaptic weights encode matrices A, B, C, L, K.
- Updates to a weight (e.g. in matrix A) use **only**:
  - Presynaptic activity (e.g., $\hat{x}_{t-\tau}$),
  - Postsynaptic activity or input current (e.g., Le_t),
  - Which are **directly available at the synapse**.

There's no need to reference faraway neurons or global variables like the full matrix $C^\top$.

❌ **The alternative ANN (red, panel C) is not biologically plausible because:**

- It must **recompute all intermediate states** from $\hat{x}_{t+1-\tau}$ to $\tilde{x}_t$, which means:
  - You need **exact copies of A and B** at each step (unrealistic duplication).
  - You need to store a **long memory of past controls** $u_{t-\tau}, \ldots, u_{t-1}$ — biologically hard.
  - This creates **weight transport** and **temporal loop dependencies**, which violate locality.

Control: Policy Gradient

3.2.2 Control

Next, we consider optimal control a neural implementation of which was missing in most previous proposals, cf. Table 1. Traditionally, optimal control law is computed by iterating a matrix Riccati equation, cf. Supplementary Material, posing a difficult challenge for a biological neural implementation (but see [21]). To circumvent this problem, we propose to learn the controller weights $\boldsymbol{K}$ using a policy gradient method [12, 22] instead. Policy gradient methods directly parametrize a stochastic controller $\pi_{\boldsymbol{K}}(\boldsymbol{u}|\boldsymbol{x})$. Representing the total cost for a given trajectory τ as $c(\tau)$, they optimize the parameters $\boldsymbol{K}$ by performing gradient descent on the expected cost $J = \mathbb{E}_{\pi_{\boldsymbol{K}}}\left[c(\tau)\right] = \mathbb{E}_{\pi_{\boldsymbol{K}}}\left[\sum_{t=0}^{T} c_t\right]$.

$$\nabla_{\boldsymbol{K}} J = \int c(\tau)\nabla\pi_{\boldsymbol{K}}(\tau)d\tau = \mathbb{E}_{\pi_{\boldsymbol{K}}}\left[c(\tau)\nabla_{\boldsymbol{K}}\log\pi_{\boldsymbol{K}}(\tau)\right] \tag{17}$$

$$= \mathbb{E}_{\pi_{\boldsymbol{K}}}\left[\left(\sum_{t=0}^{T} c_t\right)\left(\sum_{s=0}^{T}\nabla_{\boldsymbol{K}}\log\pi_{\boldsymbol{K}}(\boldsymbol{u}_s|\boldsymbol{x}_s)\right)\right] \tag{18}$$

The term in square brackets is an unbiased estimator of the gradient and can be used to perform stochastic gradient decent. As already hinted at by [12], due to causality, costs c_t are not affected by later controls $\boldsymbol{u}_s$, $s > t$, and the variance of the estimator can be reduced by excluding those terms, which yields

$$\Delta\boldsymbol{K} \propto -\sum_{t=0}^{T} c_t\left(\sum_{s=0}^{t}\nabla_{\boldsymbol{K}}\log\pi_{\boldsymbol{K}}(\boldsymbol{u}_s|\boldsymbol{x}_s)\right). \tag{19}$$

We simply keep an eligibility trace of the past, $\boldsymbol{Z}_t = \sum_{s=0}^{t}\nabla_{\boldsymbol{K}}\log\pi_{\boldsymbol{K}}(\boldsymbol{u}_s|\boldsymbol{x}_s)$, and perform parameter updates $\Delta\boldsymbol{K} \propto -c_t\boldsymbol{Z}_t$ at each time step t. A similar update rule has been suggested by [23, 24] for the infinite horizon case. Global convergence of policy gradient methods for linear quadratic regulator has been recently studied by Fazel *et al.* [25].

We assume the output of neurons encoding control $\boldsymbol{u}$ is perturbed by Gaussian noise

$$\boldsymbol{u}_t = -\boldsymbol{K}\hat{\boldsymbol{x}}_t - \boldsymbol{\xi}_t \quad \text{with} \quad \boldsymbol{\xi}_t \sim \mathcal{N}(0, \sigma^2\boldsymbol{I}). \tag{20}$$

The synapses are endowed with a synaptic tag [26] $\boldsymbol{Z}$, an eligibility trace that tracks correlations between pre-synaptic activity $\hat{\boldsymbol{x}}$ and post-synaptic noise $\boldsymbol{\xi}$. It is reset to zero at the beginning of each trajectory, though instead of a hard reset it could also softly decay with a time constant of the same order $\mathcal{O}(T)$ as trajectory duration [27]. The weight update assigns cost c_t to the synapses according to their eligibility $\boldsymbol{Z}_t$. The cost is e.g. provided by a diffuse neuromodulatory signal such as dopamine.

The optional use of momentum $m \in [0, 1)$ adds a low-pass filter to the synaptic plasticity cascade:

$$\boldsymbol{Z}_t = \boldsymbol{Z}_{t-1} + \boldsymbol{\xi}_t\hat{\boldsymbol{x}}_t^\top \quad \left(= \sigma^2\textstyle\sum_{s=0}^{t}\nabla_{\boldsymbol{K}}\log\pi_{\boldsymbol{K}}(\boldsymbol{u}_s|\hat{\boldsymbol{x}}_s)\right) \tag{21}$$

$$\boldsymbol{G}_t = m\boldsymbol{G}_{t-1} + c_t\boldsymbol{Z}_t \tag{22}$$

$$\Delta\boldsymbol{K}_t \propto -\boldsymbol{G}_t \tag{23}$$

•

📉 What's the idea behind policy gradient?

You define a **stochastic controller**:

$$\pi_K(u|x)$$

Which gives a probability distribution over actions given a state. The goal is to minimize the **expected total cost**:

$$J = \mathbb{E}_{\pi_K}[c(\tau)] = \mathbb{E}_{\pi_K}\left[\sum_{t=0}^{T} c_t\right]$$

- You then take gradients of this cost with respect to the controller parameters K and **descend along the gradient**.

The network:

1. Tries slightly noisy actions.
2. Sees how well they worked (based on cost).
3. Adjusts control weights K using **local activity correlations**, guided by **global reward/punishment**.

- This replaces the mathematically heavy Riccati solution with a biologically inspired, lightweight learning rule.

🧠 How does the network do it?

Here's how it's implemented in the brain-like network:

🌑 Step 1: Add noise to the output control signal

$$u_t = -K\hat{x}_t - \xi_t, \quad \xi_t \sim \mathcal{N}(0, \sigma^2 I)$$

This randomness is **exploration** — it lets the network see the effect of slightly different actions.

🧠 Step 2: Track correlations between:

- The **noise** ξ_t,
- The **presynaptic input** $\hat{x}_t$

This correlation is stored in an **eligibility trace** Z_t:

$$Z_t = Z_{t-1} + \xi_t \hat{x}_t^\top \tag{21}$$

This acts like a memory of how noise influenced actions over time.

💡 Step 3: Assign credit using the cost

The idea is: if the outcome (cost c_t) was bad, and noise ξ_t pushed the action a certain way, then maybe that direction was bad — so adjust K to avoid it.

Define:

$$G_t = mG_{t-1} + c_t Z_t \tag{22}$$

$$\Delta K_t \propto -G_t \tag{23}$$

This is a **local plasticity rule**: you update synaptic weights using:

- Pre-synaptic input ($\hat{x}_t$),
- Post-synaptic error signal (ξ_t),
- Global cost signal (c_t).