GATED TARGET PROPAGATION FOR COMPOSITIONAL GENERALIZATION IN CONTINUAL LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Continual learning is typically framed as acquiring new knowledge without catastrophically forgetting previous tasks. However, a flexible continual learner should also be able to reuse and recombine previously acquired knowledge to rapidly solve novel task compositions. We introduce Gated Target Propagation (GaTaP), a continual learning algorithm in which task-specific gating variables—learned through a closed-form inner loop update—selectively suppress or enhance network modules. Network parameters are learned in a slower timescale outer loop, using the same local difference target propagation error signal as is used for adapting gating variables. We provide tractable experiments on class-incremental learning scenarios for both multilayer perceptron and convolutional network architectures. We show strong performance retention on previously learned tasks, as well as compositional generalization to unseen tasks, achieved through few-shot gain adaptation at inference. We analyze learned gating patterns and find that related tasks exhibit similar gating patterns, suggesting that inferred gates capture meaningful, reusable task structure. Overall, GaTaP provides a powerful framework for jointly ameliorating catastrophic forgetting and enabling few-shot compositional generalization in neural network models.

1 INTRODUCTION

Continual learning aims to sequentially train models on new experiences while preserving knowledge acquired from previous ones. Neural networks, however, often suffer from *catastrophic forgetting*: learning a new task can overwrite parameters that are important for earlier tasks, causing their performance to deteriorate (McCloskey & Cohen, 1989; French, 1999; Goodfellow et al., 2013). A large body of work addresses this problem through regularization (Kirkpatrick et al., 2017; Zenke et al., 2017), replay (Lopez-Paz & Ranzato, 2017; Rebuffi et al., 2017; Buzzega et al., 2020), or architectural mechanisms that isolate task-specific parameters (Rusu et al., 2016; Fernando et al., 2017; Mallya & Lazebnik, 2018; Serrà et al., 2018; Veniat et al., 2021; Rajasegaran et al., 2020). While these approaches can improve retention, continual learning within these studies is typically evaluated only through the ability to remember previously learned tasks.

We argue that remembering alone is not sufficient for flexible continual learning. A capable learner should also be able to *reuse and recombine* previously acquired knowledge to solve novel tasks. For example, after learning to discriminate between MNIST (LeCun et al., 2010) digits 3 and 4, and between 5 and 6, a model should ideally be able to adapt to unseen combinations such as 3 vs. 6 or 4 vs. 5 without relearning the underlying representations. This connects continual learning with *compositional generalization*: the ability to solve novel combinations of previously encountered task components (Lake & Baroni, 2017; Elmoznino et al., 2025). From this perspective, continual learning should not only preserve knowledge, but organize it into reusable computational components that can be rapidly recombined.

We introduce **GaTaP** (**Ga**ted **Ta**rget **P**ropagation), a learning framework designed to jointly address catastrophic forgetting and compositional generalization. GaTaP represents shared knowledge through a collection of subnetworks whose contributions are modulated by task-specific multiplicative gain variables, denoted by Γ. These gains act as soft gates that activate or inactivate subnetworks depending on the current task. The same shared weights can therefore support different tasks through different configurations of Γ, providing a mechanism for selective plasticity and knowledge reuse.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Training GaTaP networks is a form of supervised meta-learning. For a given task, an inner optimization loop selects a gain vector Γ that gates subnetworks (fixed for inner loops), while an outer loop optimization updates subnetwork parameters for an ensemble of tasks, each with their own Γ. This separates *fast task adaptation* from *slow, shared-weight learning* through a bilevel optimization scheme analogous to the inference and parameter-update steps of variational expectation maximization (VEM) (Bernardo et al., 2003). Crucially, optimization is carried out with difference target propagation (Lee et al., 2015). Rather than propagating an output gradient through the entire network, target propagation infers local target states for individual layers, providing a layer-wise credit-assignment mechanism that is local (Bredenberg et al., 2023), compatible with gated modular computation, and aligned with biologically plausible learning (Bengio, 2014). The use of this local learning rule has a distinctive implication for the inner loop: **learning optimal gain parameters for a new task reduces to a series of least square problems which can be solved in closed form, enabling adaptation at inference without gradient descent**.

The compositional generalization capabilities of GaTaP thus manifest at *inference time*, once the outer loop training is complete and all shared weights are frozen. In this setting, adaptation to either previously learned or novel tasks is performed solely by inferring a new gain configuration Γ from a small support set (a simple closed-form evaluation). Consequently, success on an unseen task directly tests whether the model has acquired reusable computational components: no modification of the shared representation is allowed during adaptation.

Our main contributions are:

- We formulate continual learning as a joint problem of *retention and composition*, requiring learned knowledge to remain both stable and reusable for unseen task combinations.
- We introduce GaTaP, which separates fast, task-specific routing through sparse gain inference from slower, shared representation learning through target propagation.
- We derive a closed-form, sparse update for the task-specific gains and a probabilistic target-inference rule that extends from multilayer perceptron to convolutional architectures.
- We show experimentally that GaTaP retains previously learned tasks while adapting to unseen task compositions with frozen shared weights, and that related tasks exhibit similar inferred gating patterns.

2 Related Work

Continual learning. Continual-learning methods are commonly grouped into regularization-, replay-, and architecture-based approaches. Regularization methods constrain updates to parameters considered important for previous tasks, as in Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017) and Synaptic Intelligence (SI) (Zenke et al., 2017). Replay-based approaches instead revisit stored or generated examples from previous tasks, including GEM (Lopez-Paz & Ranzato, 2017), A-GEM (Chaudhry et al., 2018), iCaRL (Rebuffi et al., 2017), and Dark Experience Replay (Buzzega et al., 2020). While effective at mitigating catastrophic forgetting, these approaches primarily preserve previous knowledge rather than explicitly organizing it into computational components that can be recombined for novel tasks.

Modular networks and task-dependent routing. Architectural approaches reduce interference by assigning different parameters or computational pathways to different tasks. Progressive Neural Networks (Rusu et al., 2016) add and freeze task-specific columns, whereas PathNet (Fernando et al., 2017), PackNet (Mallya & Lazebnik, 2018), and Hard Attention to the Task (Serrà et al., 2018) select, reserve, or protect subsets of a larger network. In many such approaches, catastrophic forgetting is mitigated partly by construction: previously allocated parameters are frozen or future updates are restricted according to a predefined routing or capacity-allocation mechanism. While effective at reducing interference, this shifts an important part of the learning problem—determining which components are relevant to a new task and should therefore be reused or protected—into the architecture or training rules.

Particularly related to our work, RPS-Net (Rajasegaran et al., 2020) progressively searches for task-specific paths through a modular network while encouraging the reuse of previously selected paths, thereby enabling forward transfer and reducing interference. Veniat et al. (Veniat et al., 2021) further

propose Modular Networks with Task-Driven Priors (MNTDP), in which tasks are represented as compositions of reusable modules. A data-driven prior restricts architecture search to perturbations of paths associated with related previous tasks, enabling efficient transfer and improved scaling to long task sequences.

GaTaP shares the motivation of modular reuse but treats pathway selection itself as an inference problem. Rather than prescribing a discrete architecture or a fixed rule specifying which subnetworks should be protected or recruited, GaTaP represents task relevance through sparse multiplicative gains Γ inferred directly from a support set. These gains determine which shared subnetworks contribute to the current task and, consequently, which components participate strongly in learning. Thus, the routing and selective-plasticity rules are encoded in learned task-dependent variables rather than imposed externally. At test time, the same mechanism allows new gain configurations to be inferred for previously unseen task compositions while all shared weights remain frozen.

Compositional generalization. Compositional generalization concerns the ability to systematically recombine previously learned components to solve novel task combinations (Zhou et al., 2024; Lake & Baroni, 2017; Lake et al., 2015). Although modular continual-learning methods naturally promote parameter reuse, transfer to a new task does not necessarily imply compositional generalization (Mittal et al., 2022; 2025). We explicitly evaluate whether components acquired during sequential learning can be recombined without modifying the shared representation. For example, after learning to discriminate between MNIST (LeCun et al., 2010) digits 3 and 4, and between digits 5 and 6, GaTaP is evaluated on unseen combinations such as 3 vs. 6 by inferring only a new gating configuration. This provides a direct test of whether continual learning has produced reusable computational structure rather than merely preserved task-specific solutions.

Target propagation and local credit assignment. Target propagation provides an alternative to standard backpropagation by assigning desired activation states, or *targets*, to intermediate layers (Bengio, 2014; Lee et al., 2015). Difference Target Propagation (Lee et al., 2015), for instance, propagates corrected activation targets through learned approximate inverses rather than propagating only loss gradients. GaTaP builds on difference target propagation by deriving hidden targets through local Gaussian conditioning under a linearized network model, yielding Kalman-gain updates related to Gauss-Newton target methods (Meulemans et al., 2020).

3 Gated Target Propagation

GaTaP separates continual learning into two coupled processes operating at different timescales: *fast task-specific routing* (inner loop), controlled by multiplicative gains Γ, and *slow shared representation learning* (outer loop), stored in weights W. Given a task $\mathcal{T}$, a support set (training examples) is used to infer a sparse gating configuration $\Gamma^\star_{\mathcal{T}}$, while a disjoint query set is used to update the shared subnetworks with the inferred gates fixed. At inference time, W is frozen and adaptation to a task occurs only through re-inference of Γ. Learning a task can thus happen in two ways: either through shaping shared weights W during dual phase training, or through gain adaptation alone at inference.

3.1 Gated probabilistic network

Consider an L-layer network with hidden states $h_i \in \mathbb{R}^{d_i}$, where $h_0 = x$ denotes the input. Rather than using a single transformation at layer i, GaTaP contains n_i shared subnetworks $\{W_i^{(k)}\}_{k=1}^{n_i}$. Their contributions are controlled by task-specific multiplicative gains $\Gamma_i^{(k)} \in \mathbb{R}^{d_i}$:

$$h_i = \sum_{k=1}^{n_i} \Gamma_i^{(k)} \odot \Big(W_i^{(k)} \phi_{i-1}(h_{i-1}) \Big) + \epsilon_i, \qquad \epsilon_i \sim \mathcal{N}(0, \Sigma_i), \tag{1}$$

where ϕ_{i-1} is the activation function and $\odot$ denotes elementwise multiplication. Thus, $\Gamma_i^{(k)}$ gates the output units of the k-th subnetwork. We define the corresponding task-dependent effective map as

$$W_i(\Gamma_i) = \sum_{k=1}^{n_i} \mathrm{Diag}\Big(\Gamma_i^{(k)}\Big)\, W_i^{(k)}, \tag{2}$$

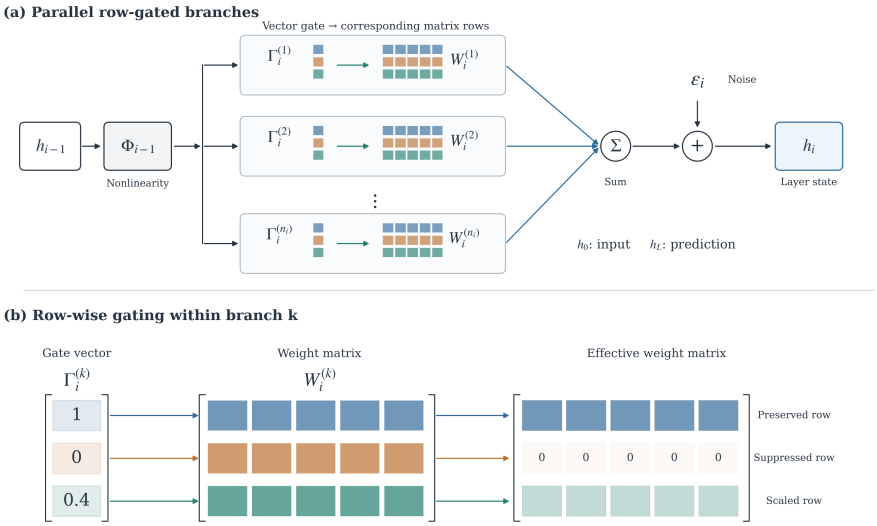


Figure 1: **Parallel subnetworks with task-specific row-wise gating.** (a) The transformed state $\Phi_{i-1}(h_{i-1})$ is processed by n_i parallel subnetworks with shared weight matrices $W_i^{(k)}$. Each task-specific gate vector $\Gamma_i^{(k)}$ scales the rows of its corresponding weight matrix. The resulting branch outputs are summed, and additive noise ϵ_i yields the next layer state h_i. (b) Illustration of row-wise gating within one subnetwork: each gate entry multiplies every weight in the corresponding row. Values of 1, 0, and 0.4 respectively preserve, suppress, and attenuate that row, allowing task-dependent modulation of shared connectivity.

so that

$$h_i = W_i(\Gamma_i)\phi_{i-1}(h_{i-1}) + \epsilon_i. \tag{3}$$

The subnetworks $W_i^{(k)}$ are shared across tasks, whereas the gains $\Gamma_i^{(k)}$ are inferred separately for each task. A gain close to zero suppresses the contribution of the corresponding unit or subnetwork, thereby reducing its participation both in the forward computation and in subsequent learning.

We interpret Equation 3 probabilistically as

$$p(h_i \mid h_{i-1}, \Gamma_i, W_i) = \mathcal{N}\left(W_i(\Gamma_i)\phi_{i-1}(h_{i-1}), \Sigma_i\right). \tag{4}$$

Under this model, hidden states and gains are latent variables that can be inferred using both the input and the desired output. This provides the probabilistic basis for target propagation: rather than communicating only a gradient from the output loss, we infer corrected hidden states $\{h_i^\star\}_{i=1}^L$ and use them as local learning targets. A Laplace prior on the gains, $p(\Gamma_i) \propto \exp(\alpha_i \|\Gamma_i\|_1)$, encourages sparse task-dependent routing. Full derivations of the corresponding variational objective are provided in Appendix A.1.1.

3.2 Probabilistic target inference

Target propagation requires a desired state $h_i^\star$ for each intermediate layer. GaTaP obtains these targets through approximate probabilistic inference. Consider the forward mapping from layer i to $i+1$,

$$F_{i+1}(h_i) = W_{i+1}(\Gamma_{i+1})\phi_i(h_i), \qquad h_{i+1} = F_{i+1}(h_i) + \epsilon_{i+1}. \tag{5}$$

For a linear-Gaussian mapping, the conditional posterior over h_i given a desired state at the layer above is Gaussian, and its posterior mean can be computed analytically using a Kalman update. For nonlinear networks, we locally linearize F_{i+1} around the current feedforward state h_i. Let

$$J_{i+1} = \left.\frac{\partial F_{i+1}(h)}{\partial h}\right|_{h=h_i} \tag{6}$$

denote the local Jacobian. The resulting extended-Kalman target update is:

$$h_i^\star = h_i + K_i \left(h_{i+1}^\star - h_{i+1} \right), \tag{7}$$

with

$$K_i = \Sigma_i J_{i+1}^\top \left(J_{i+1} \Sigma_i J_{i+1}^\top + \Sigma_{i+1} \right)^{-1}. \tag{8}$$

The output target is initialized from the prediction error,

$$h_L^\star = h_L + K_L(y - h_L), \tag{9}$$

and Equation 7 is then applied recursively from the output toward earlier layers.

Equation 7 has a simple interpretation: the inferred target is the current feedforward state plus a correction determined by the discrepancy at the next layer. The gain matrix K_i controls how strongly this discrepancy should modify h_i according to the uncertainty assigned to adjacent layers. Unlike standard backpropagation, the signal delivered to layer i is therefore a target state $h_i^\star$ rather than only the gradient of a global loss. The linear-Gaussian derivation along with its nonlinear extension are given in Appendices A.1.2 and A.2.

3.3 Fast, sparse gate inference

Given the target states, GaTaP infers which shared subnetworks should be recruited for the current training epoch, without explicit knowledge of which task is being performed. Importantly, this adaptation modifies only Γ, while the shared weights remain fixed.

Let $\mathcal{S}_\mathcal{T} = \{(x_p, y_p)\}_{p=1}^P$ be the support set for a given training epoch. For layer i, define the response of subnetwork k to support example p as

$$Z_{i,p}^{(k)} = W_i^{(k)} \phi_{i-1}(h_{i-1,p}). \tag{10}$$

The gated prediction of the target state is:

$$h_{i,p} = \sum_{k=1}^{n_i} \Gamma_i^{(k)} \odot Z_{i,p}^{(k)}. \tag{11}$$

We infer the gains by fitting the locally inferred targets while imposing an ℓ_1 penalty:

$$\mathcal{L}_{i,\text{support}}(\Gamma_i) = \frac{1}{2P} \sum_{p=1}^{P} \left\| h_{i,p}^\star - \sum_{k=1}^{n_i} \Gamma_i^{(k)} \odot Z_{i,p}^{(k)} - \epsilon_i \right\|_2^2 + \alpha_i \sum_{k=1}^{n_i} \|\Gamma_i^{(k)}\|_1. \tag{12}$$

The sparsity penalty corresponds to a Laplace prior over Γ and favors task-specific solutions that recruit only a subset of the available components.

Although Equation 12 couples the gains of multiple subnetworks, coordinate descent reduces each update to a scalar ℓ_1 regularized least-squares problem. For neuron q of subnetwork k, define the partial residual:

$$R_{i,p,q}^{(-k)} = h_{i,p,q}^\star - \sum_{\ell \neq k} \gamma_{i,q}^{(\ell)} Z_{i,p,q}^{(\ell)} - \epsilon_i, \tag{13}$$

where $\gamma_{i,q}^{(\ell)} \in \Gamma_i^{(\ell)}$ is the gain variable specific to neuron q for subnetwork ℓ. Further, define the sufficient statistics:

$$a_{i,k,q} = \frac{1}{P} \sum_{p=1}^{P} \left(Z_{i,p,q}^{(k)} \right)^2, \qquad b_{i,k,q} = \frac{1}{P} \sum_{p=1}^{P} Z_{i,p,q}^{(k)} R_{i,p,q}^{(-k)}. \tag{14}$$

Given these, the coordinate-wise optimum has the closed form:

$$\boxed{\left(\gamma_{i,q}^{(k)} \right)^\star = \text{sign}(b_{i,k,q}) \times \frac{\lfloor |b_{i,k,q}| - \alpha_i \rfloor^+}{a_{i,k,q}},} \tag{15}$$

where $\lfloor z \rfloor^+ = \max(z, 0)$. Thus, components whose support-set evidence does not exceed the sparsity threshold are assigned exactly zero gain. The resulting $\Gamma^\star$ therefore defines a sparse computational route specific to the selected task for that training epoch, while blocking learning signal flow backward through inactive pathways during learning. The complete derivation can be found in Appendix A.3.1.

3.4 SHARED-WEIGHT LEARNING

After inferring $\Gamma^\star$ from the support set, the gains are held fixed and the shared weights are optimized on a disjoint query set $\mathcal{Q}$. For each query example, target states are first computed using Equations 7-8. The local objective at layer i is

$$\mathcal{L}_{i,\text{query}} = \frac{1}{2|\mathcal{Q}_\mathcal{T}|} \sum_{(x,y)\in\mathcal{Q}_\mathcal{T}} \left\| h_i^\star - \sum_{k=1}^{n_i} \left(\Gamma_i^{(k)}\right)^\star \odot W_i^{(k)} \phi_{i-1}(h_{i-1}) - \epsilon_i \right\|_{\Sigma_i^{-1}}^2, \tag{16}$$

where $\|v\|_{\Sigma^{-1}}^2 = v^\top \Sigma^{-1} v$. Only the shared parameters are updated:

$$W_i^{(k)} \leftarrow W_i^{(k)} - \eta \nabla_{W_i^{(k)}} \mathcal{L}_{i,\text{query}}. \tag{17}$$

Training can therefore be summarized as the bi-level procedure:

$$\Gamma^\star(W) = \arg\min_\Gamma \left[\mathcal{L}_{\text{support}}(\Gamma; W, \mathcal{T}) + \sum_{i,k} \alpha_i \|\Gamma_i^{(k)}\|_1 \right], \tag{18}$$

followed by

$$W \leftarrow W - \eta \nabla_W \mathcal{L}_{\text{query}} \left(W, \Gamma_\mathcal{T}^\star(W); \mathcal{T}\right). \tag{19}$$

The two stages play distinct roles: the inner problem rapidly determines which shared components should be recruited for the current task, while the outer problem updates the associated shared weights. Because the same shared parameters are reused across tasks, this decomposition allows task-specific computation without maintaining an independent network for every task. Refer to Algorithm 1 for the full training procedure.

Inference on seen and unseen tasks. After the complete continual-learning sequence, the shared parameters are frozen:

$$W \leftarrow W^\star, \qquad \Delta W = 0. \tag{20}$$

For an evaluation task $\mathcal{T}$, either previously seen or an unseen composition, only $\Gamma^\star$ is computed according to Equation 18. Performance is then measured on a disjoint query set. Consequently, performance on an unseen task cannot be attributed to further learning of the shared representation. Instead, the model rapidly reconfigures previously acquired components through few-shot inference of $\Gamma^\star$.

3.5 CONVOLUTIONAL EXTENSION

To scale GaTaP beyond fully connected networks, we extend the same gating and target-propagation mechanism to convolutional architectures by replacing each linear subnetwork response with a convolution. For feature map h_{i-1},

$$h_i = R_i\left(h_{i-1}\right) + \sum_{k=1}^{n_i} \Gamma_i^{(k)} \odot \left(W_i^{(k)} * \phi_{i-1}(h_{i-1}) \right) + \epsilon_i, \tag{21}$$

where $*$ denotes the convolution operation and R_i is the residual connection. Additionally, gains can be applied channel-wise, using one gain per output channel shared across all spatial locations, or spatially, using location-specific gains that modulate different positions within each feature map. The sparse coordinate update in Equation 15 remains unchanged in form, with its sufficient statistics additionally aggregated over spatial positions for channel-wise gating. The target update also retains the form of Equation 7. In convolutional networks, the Kalman gain becomes computationally expensive due to the large matrix inversion in $(J_i \Sigma_i J_i^\top + \Sigma_{i+1})^{-1}$. We therefore use a diagonal approximation in our convolutional experiments that renders inversion trivial. Further details are provided in Appendix A.4.

4 EXPERIMENTAL SETUP

Our experiments are designed to test the two central claims of the proposed framework: whether GaTaP can retain knowledge acquired sequentially, and whether the resulting shared components can be recombined to solve novel task compositions. We therefore focus on controlled class-incremental classification experiments in which both retention and compositional reuse can be measured directly.

Datasets and architectures. We evaluate GaTaP on MNIST, CIFAR-10, and CIFAR-100, progressively increasing the visual complexity and number of classes. MNIST experiments use a feed-forward gated network, whereas CIFAR-10 and CIFAR-100 use convolutional architectures. The main experiments are formulated as sequences of binary classification tasks. Additional preliminary multi-class experiments are reported in Supplemental Figure S3; these results are intended as an initial test of scalability beyond the binary setting, and a more systematic multi-class evaluation is left for future work. Architecture and hyperparameter details are provided in Appendix A.5.

Class-incremental task construction. All main experiments use the class-incremental learning setting: tasks are presented sequentially, and the original class identities are preserved throughout learning. For example, an MNIST training stream may sequentially require discrimination between digits 1 and 2, 3 and 4, 5 and 6, and 7 and 8. The model has access only to the current task data during its corresponding training stage. We distinguish between *seen tasks*, which are presented during sequential training, and *unseen compositional tasks*, which recombine classes encountered in different training tasks. For instance, after learning 1 vs. 2 and 3 vs. 4, tasks such as 1 vs. 4 or 2 vs. 3 require a new combination of previously acquired components.

Training and evaluation protocol. For each task $\mathcal{T}_j$, its data are divided into a support set $\mathcal{S}_j$ and a disjoint query set $\mathcal{Q}_j$. During training, $\mathcal{S}_j$ is used to infer the task-specific gains $\Gamma_j^\star$, while $\mathcal{Q}_j$ is used to update the shared weights W through target propagation. After the complete task sequence, all shared weights are frozen. Evaluation on both seen and unseen tasks is then performed by inferring only a new gain configuration from the task support set and measuring performance on its query set. Consequently, performance on unseen compositions cannot be attributed to further learning of the shared representation.

Evaluation criteria. Retention is measured by the final accuracy on previously seen tasks. Compositional generalization is measured by query accuracy on unseen class combinations with the shared weights fixed. We additionally analyze the cosine similarity between inferred gain vectors to determine whether tasks sharing classes recruit similar computational routes, and vary the support-set size to evaluate few-shot inference of gain variables. GaTaP-trained models were compared to equivalent architectures trained with backpropagation and EWC (with code adapted from (Chua et al., 2026)).

5 RESULTS

Retention and compositional generalization. After training, all shared weights W were frozen; only the task-specific gains $\Gamma_{\mathcal{T}}$ were inferred from a support set, and performance was measured on a disjoint query set. On seen tasks (Fig. 2a), GaTaP reached 0.978 final average accuracy on MNIST, compared to 0.301 (Backprop) and 0.456 (EWC); similar gaps held on CIFAR-10. This indicates that GaTaP effectively protects learned representations compared to alternative methods. The same frozen representation also generalized to unseen task combinations (Fig. 2.b): with accuracy reaching 0.956 on unseen digit combinations and 0.851 on a held-out task never seen during training (MNIST), and 0.895 / 0.643 on CIFAR-10. Since W was never updated at evaluation time, this adaptation came solely from inferring a new gain configuration, via a single closed-form optimization step, using 64 support examples; Fig. 2d shows comparable accuracy with far fewer examples shown. This supports the central claim that the learned components can be recombined for few-shot generalization through a single, close-form optimization step.

Structure of the inferred gating configurations. We next investigated whether the inferred task-specific gains encoded meaningful relationships between tasks. If the gains identified reusable computational components, then tasks sharing classes should be expected to recruit more similar routes than unrelated tasks. We therefore computed the cosine similarity between the final gain configurations of different task pairs and grouped them according to their degree of class overlap. As shown in Fig. 3, Γ similarity tracked task overlap closely on MNIST (near-identical for matching tasks, near-zero for disjoint ones), and a similar trend was observed on CIFAR-10. This indicates that related tasks reuse related computational pathways, while unrelated tasks recruit more distinct ones.

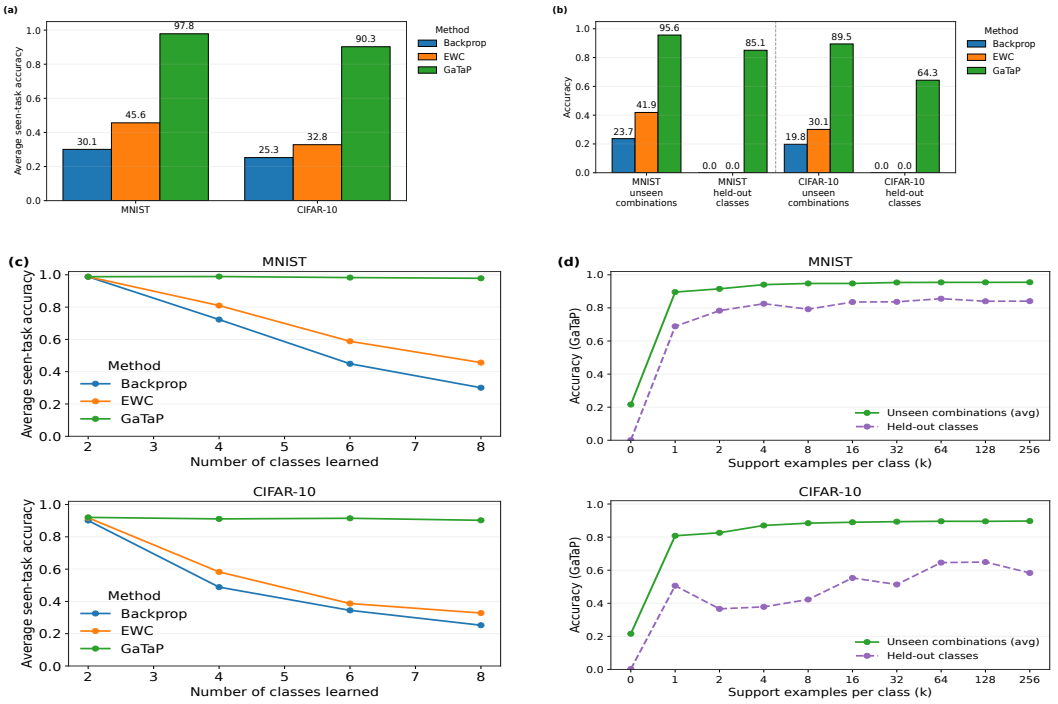


Figure 2: MNIST vs. CIFAR-10 comparison across methods. **a)** Average performance on tasks networks have already been exposed to during the training process, evaluated at the end of training. **b)** Average performance on unseen task combinations of data classes already seen during training, compared to performance on task combinations never seen during training. **c)** Same as (a), evaluated as networks are trained on an increasing number of tasks for MNIST (top) and CIFAR-10 (bottom). **d)** Average performance on unseen task combinations as a function of the number of support set examples provided per-class to train gates Γ during evaluation, for MNIST (top) and CIFAR-10 (bottom).

6 Discussion and Limitations

In this study we have introduced *gated target propagation* (GaTaP), a learning algorithm capable of both preventing catastrophic forgeting and enabling compositional generalization through a shared mechanism: the rapid optimization of network gains Γ across epochs. Our method does not require explicit knowledge of task transitions—only output errors—and it outperforms both standard back propagation and EWC on continual learning benchmarks, while simultaneously supporting few-shot compositional generalization to task recombinations and wholly unseen classification tasks.

Our method can be seen as a generalization of difference target propagation (Lee et al., 2015), as well as other methods that use Kalman-style error corrections to provide closed-loop modification of internal network states prior to learning. Previous work, generally restricted to *additive*—as opposed to our *multiplicative*—corrections, have shown that this form of closed-loop control can provide benefits for learning stability, particularly in chaotic dynamical systems (Sağtekin et al., 2026; Kaleb et al., 2024; Sussillo & Abbott, 2009). Our work builds on these methods by showing how similar error-based feedback signals can be used in concert with gain modulation mechanisms to enable continual learning and compositional generalization. Collectively, these studies demonstrate that error signals used for both control and learning can provide important improvements relative to standard backpropagation-based methods.

Interestingly, methods closely related to GaTaP, such as difference target propagation and control-theoretic learning methods, have been proposed as models of learning and synaptic plasticity in the brain (Podlaski & Machens, 2020; Gilra & Gerstner, 2017; Alemi et al., 2018). While our method relies on biologically implausible operations to compute feedback errors (e.g. calculating a Kalman gain matrix), previous work has demonstrated that feedback projections can still enable learning

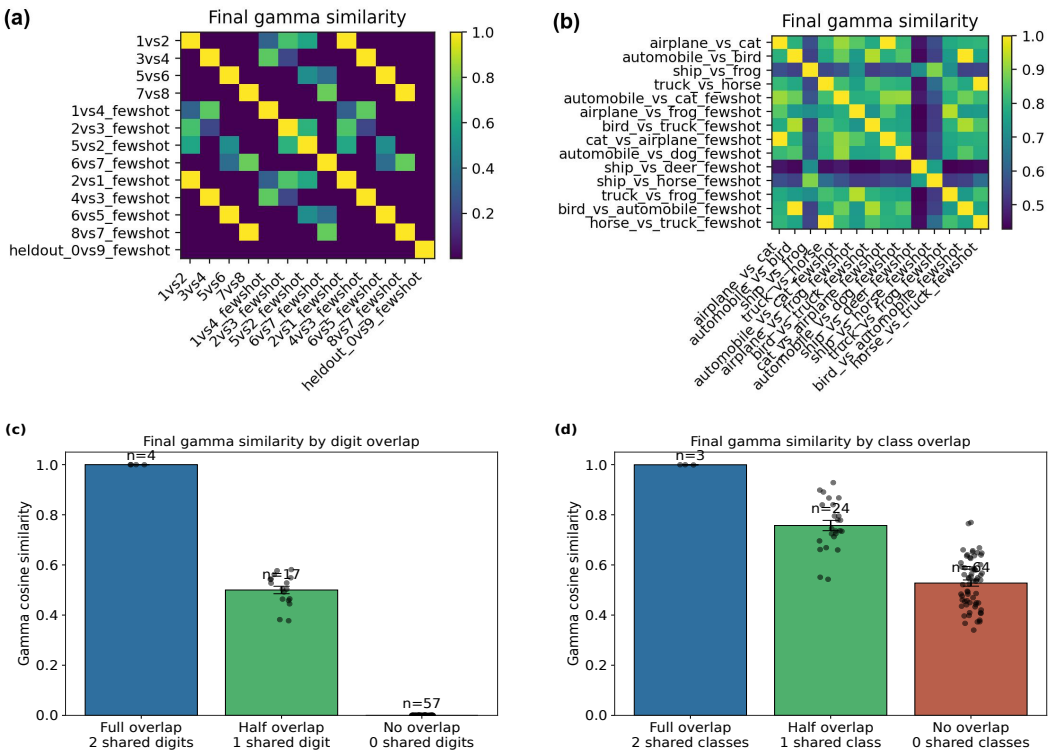


Figure 3: Visualizing gain similarity across tasks. **a)** Cosine similarity of learned Γ parameters across MNIST two-way classification tasks. "fewshot" indicates an unseen recombination of previously seen digits. **b)** Same as (a), for CIFAR-10 two-way classification tasks. **c)** Average cosine similarity of Γ parameters for MNIST two-way classification tasks that share both digits, one digit, or no digits. **d)** Same as (c), for CIFAR-10 two-way classification tasks.

when they adapt through local plasticity to dynamically approximate the analytically optimal feedback signal (Lee et al., 2015; Meulemans et al., 2021). Therefore, relating GaTaP more concretely to models of learning in the brain remains a promising direction for future work.

The present study nevertheless has several limitations. First, computing the Kalman-style target update involves Jacobian–covariance operations that become expensive in larger networks. Our convolutional network implementation uses a diagonal approximation (Appendix A.4) to make this computation tractable, introducing a trade-off between inference fidelity and computational efficiency. Methods to approximate or amortize the computation of feedback errors may greatly increase the computational efficiency of our method. Finally, the current experiments focus on static visual tasks. Extending the framework to temporal models, larger architectures, and more complex task compositions will be important for assessing its generality.

7 Conclusion

We have introduced GaTaP, a continual-learning framework that jointly addresses knowledge retention and compositional reuse. The method separates fast, task-specific adaptation from slower, shared learning: sparse, multiplicative gains infer which subnetworks are relevant to a task, while target propagation updates the shared weights through local target states. After continual training, the shared weights remain fixed and both previously learned and unseen task compositions are solved through gain inference alone, a simple closed form update. Experiments on MNIST, CIFAR-10, and CIFAR-100 show that GaTaP retains previously acquired knowledge, recombines learned components for novel tasks, and produces gating patterns that reflect task structure. These results suggest that continual learning and compositional generalization can be jointly enabled through a shared learning architecture based on rapidly reconfiguring multiplicative gains.

AI USE STATEMENT

In this work, we used generative AI tools to generate images of our architecture.

REFERENCES

Alireza Alemi, Christian Machens, Sophie Deneve, and Jean-Jacques Slotine. Learning nonlinear dynamics in efficient, balanced spiking networks using local plasticity rules. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Yoshua Bengio. How auto-encoders could provide credit assignment in deep networks via target propagation. *arXiv preprint arXiv:1407.7906*, 2014.

JM Bernardo, MJ Bayarri, JO Berger, AP Dawid, D Heckerman, AFM Smith, M West, et al. The variational bayesian em algorithm for incomplete data: with application to scoring graphical model structures. *Bayesian statistics*, 7(453-464):210, 2003.

Colin Bredenberg, Ezekiel Williams, Cristina Savin, Blake Aaron Richards, and Guillaume Lajoie. Formalizing locality for normative synaptic plasticity models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=rGN3X9jnEg.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: A strong, simple baseline. In *Advances in Neural Information Processing Systems*, 2020.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

Raymond Chua, Doina Precup, and Blake Aaron Richards. Balancing plasticity and stability with fast and slow successor features. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=SbOzVVh3kr.

Eric Elmoznino, Thomas Jiralerspong, Yoshua Bengio, and Guillaume Lajoie. Towards a Formal Theory of Representational Compositionality. June 2025. URL https://openreview.net/forum?id=fXCfT7ErvL.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A. Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999.

Aditya Gilra and Wulfram Gerstner. Predicting non-linear dynamics by stable local learning in a recurrent spiking neural network. *Elife*, 6:e28295, 2017.

Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Klara Kaleb, Barbara Feulner, Juan A Gallego, and Claudia Clopath. Feedback control guides credit assignment in recurrent neural networks. *Advances in Neural Information Processing Systems*, 37:5122–5144, 2024.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

Brenden M. Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. *arXiv preprint arXiv:1711.00350*, 2017.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Yann LeCun, Corinna Cortes, Chris Burges, et al. Mnist handwritten digit database. 2010.

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 498–515, 2015.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, 2017.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*, volume 24, pp. 109–165. Academic Press, 1989.

Alexander Meulemans, Francesco Carzaniga, Johan Suykens, João Sacramento, and Benjamin F. Grewe. A theoretical framework for target propagation. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 20024–20036. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/e7a425c6ece20cbc9056f98699b53c6f-Paper.pdf.

Alexander Meulemans, Matilde Tristany Farinha, Javier García Ordóñez, Pau Vilimelis Aceituno, João Sacramento, and Benjamin F Grewe. Credit assignment in neural networks through deep feedback control. *Advances in Neural Information Processing Systems*, 34:4674–4687, 2021.

Sarthak Mittal, Yoshua Bengio, and Guillaume Lajoie. Is a Modular Architecture Enough? October 2022. URL https://openreview.net/forum?id=3-3XMModtrx.

Sarthak Mittal, Eric Elmoznino, Leo Gagnon, Sangnie Bhardwaj, Guillaume Lajoie, and Dhanya Sridhar. Does learning the right latent variables necessarily improve in-context learning? In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 44504–44530. PMLR, October 2025. URL https://proceedings.mlr.press/v267/mittal25a.html.

William F Podlaski and Christian K Machens. Biological credit assignment through dynamic inversion of feedforward networks. *Advances in Neural Information Processing Systems*, 33:10065–10076, 2020.

Jathushan Rajasegaran, Munawar Hayat, Salman Khan, Fahad Shahbaz Khan, Ling Shao, and Ming-Hsuan Yang. An adaptive random path selection approach for incremental learning, 2020. URL https://arxiv.org/abs/1906.01120.

Rajesh PN Rao and Dana H Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2(1):79–87, 1999.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Maria Isabel Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties. *Institute for Systems and Robotics*, 43(46):3736–3741, 2004.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

A Sağtekin, Colin Bredenberg, and Cristina Savin. Error forcing in recurrent neural networks. *Advances in Neural Information Processing Systems*, 38:54985–55014, 2026.

Joan Serrà, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pp. 4548–4557, 2018.

David Sussillo and Larry F Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–557, 2009.

Tom Veniat, Ludovic Denoyer, and Marc'Aurelio Ranzato. Efficient continual learning with modular networks and task-driven priors, 2021. URL https://arxiv.org/abs/2012.12631.

James CR Whittington and Rafal Bogacz. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural computation*, 29(5): 1229–1262, 2017.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pp. 3987–3995, 2017.

Yanli Zhou, Reuben Feinman, and Brenden M. Lake. Compositional diversity in visual concept learning. *Cognition*, 244:105711, 2024. ISSN 0010-0277. doi: https://doi.org/10.1016/j.cognition.2023.105711. URL https://www.sciencedirect.com/science/article/pii/S0010027723003451.

A Appendix

A.1 One Hidden Layer Probabilistic Model

In this section, we describe the proposed gated target propagation framework starting from a single layer neural network model, before progressing to deeper and convolutional models. The main idea is to separate fast task adaptation from slow weight learning within a unified and theoretically justifiable framework.

We first consider a neural network with a single hidden layer. Let $s \in \mathbb{R}^{d_0}$ denote the input stimulus, $r \in \mathbb{R}^{d_1}$ the hidden state, and $o \in \mathbb{R}^{d_2}$ the output. The input and output weights are denoted by W_1 and W_2, respectively. We also include task-dependent gain variables Γ_1 and Γ_2. These variables multiplicatively modulate the activity of the network. In our model, the gain parameters act as a gating mechanism: they select which hidden units or subnetworks are active for a given task. This will allow the network to reuse shared parameters while adapting rapidly to different tasks.

For a single subnetwork per layer, the stochastic neuron equations are

$$r = \Gamma_1 \odot (W_1 s) + \epsilon_1, \qquad \epsilon_1 \sim \mathcal{N}(0, \sigma_1^2 I), \tag{22}$$

and

$$o = \Gamma_2 \odot (W_2 r) + \epsilon_2, \qquad \epsilon_2 \sim \mathcal{N}(0, \sigma_2^2 I). \tag{23}$$

where $\odot$ is the Hadamard (element-wise) product. Equivalently, we can define a conditional model probability distribution P_M that defines joint distribution of hidden states r and outputs o conditioned on Γ and W parameters:

$$P_M(r \mid s, \Gamma, W) \sim \mathcal{N}\left(\Gamma_1 \odot (W_1 s), \sigma_1^2 I\right), \tag{24}$$

and

$$P_M(o \mid r, s, \Gamma, W) \sim \mathcal{N}\left(\Gamma_2 \odot (W_2 r), \sigma_2^2 I\right). \tag{25}$$

We refer to W_1 and W_2 as subnetworks or connection weights between two consecutive layers. In a more general subnetwork formulation, two consecutive layers are connected by multiple subnetworks. For instance, $W_i^{(k)}$ with $k = 1, \ldots, n_i$, are the connections weights between layer $i-1$ and layer i. Note that in this framework there are considerably fewer gain parameters than there are weight parameters: for the hidden layer, $\Gamma_1 \in \mathbb{R}^{d_1}$, whereas $W_1 \in \mathbb{R}^{d_2} \times \mathbb{R}^{d_1}$. We next define the effective weights as follows:

$$\overline{W}_1(\Gamma_1) = \sum_{k=1}^{n_1} \text{diag}\left(\Gamma_1^{(k)}\right) W_1^{(k)}, \qquad \overline{W}_2(\Gamma_2) = \sum_{k=1}^{n_2} \text{diag}\left(\Gamma_2^{(k)}\right) W_2^{(k)}. \tag{26}$$

The neuron equations can then be written as

$$r = \overline{W}_1(\Gamma_1)\, s + \epsilon_1, \tag{27}$$

and

$$o = \overline{W}_2(\Gamma_2)\, r + \epsilon_2. \tag{28}$$

Note that if the gain for a particular unit r_j is zero ($\Gamma_{1,j} = 0$), then the effective weights for that unit $\overline{W}_{1,j,:}$ will also be zero. We will be adapting gain parameters rapidly on a per-task basis, while the weights will adapt on a slower timescale. By encouraging sparsity in our gain parameters for each task, we will in turn 'protect' weight parameters from parameter updates on tasks where they are less relevant for task performance.

A.1.1 Objective function

Let's consider a shallow neural network, where $\overline{W}_1(\Gamma_1)$ and $\overline{W}_2(\Gamma_2)$ are the effective weights:

$$\overline{W}_1(\Gamma_1) = \sum_{k=1}^{n_1} \text{diag}\left(\Gamma_1^{(k)}\right) W_1^{(k)} \text{ and } \overline{W}_2(\Gamma_2) = \sum_{k=1}^{n_2} \text{diag}\left(\Gamma_2^{(k)}\right) W_2^{(k)}. \tag{29}$$

Let $s \in \mathbb{R}^{d_0}$ be the input stimuli, r the hidden state, and o the output, governed by Equation 27 and Equation 28, respectively. For $\Gamma = (\Gamma_1, \Gamma_2)$, we assume $P_M(\Gamma)$ (marginalizing across tasks) is Laplace-distributed, which promotes sparsity for individual tasks. We further assume, for each task $\mathcal{T}$, that the data pairs (s, o) are generated from a joint data distribution $P_{\mathcal{D}}(s, o|\mathcal{T})$, which in practice correspond to input and classification target pairs. Our goal is to learn a model distribution $P_M(o|s)$ that matches the conditional data distribution $P_{\mathcal{D}}(o \mid s, \mathcal{T})$, where here we suppress dependence of P_M on the network parameters $W = (W_1, W_2)$ for conciseness and assume that the model does not have direct access to the task $\mathcal{T}$ that is being performed. Ideally, we would want to minimize the objective function:

$$\mathcal{J} = \mathbb{E}_{s,\mathcal{T}} \left[D_{KL}[P_{\mathcal{D}}(o|s, \mathcal{T})||P_M(o|s)] \right], \tag{30}$$

which would simply correspond to matching network inputs and outputs to labelled pairs, averaged across tasks and stimuli sampled from the data distribution. However, at this point, for our neural network we have only defined the conditional distribution $P_M(o, r|s, \Gamma)$, whereas the loss is defined in terms of the marginal distribution over r and Γ:

$$P_M(o|s) = \iint P_M(o, r|s, \Gamma) P_M(\Gamma) dr d\Gamma. \tag{31}$$

Computing this probability requires marginalizing over the latent gain variables Γ and the hidden state r to make $P_M(o|s)$. Since this marginalization is in practice analytically intractable, we will instead resort to minimizing a well-known upper bound on our loss, known as the *variational free energy* (Bernardo et al., 2003), which will prove much easier to compute.

To do this, we first introduce a simplified variational distribution $Q(r, \Gamma|s, o)$ to approximate the true posterior $P_M(r, \Gamma|s, o)$. Essentially, this posterior provides a correction to the network latent states when both stimuli *and* outputs have been observed. We use our variational distribution to define the variational free energy as:

$$\mathcal{L} = \mathbb{E}_{s,\mathcal{T}} \Big[D_{KL} \Big[P_{\mathcal{D}}(o|s, \mathcal{T}) Q(r, \Gamma|s, o) || P_M(o, r, \Gamma|s) \Big] \Big]. \tag{32}$$

To see that $\mathcal{L}$ is an upper bound for $\mathcal{J}$, observe:

$$\begin{aligned}
\mathcal{L} &= \mathbb{E}_{s,\mathcal{T}} \Big[D_{KL} \Big[P_{\mathcal{D}}(o|s, \mathcal{T}) Q(r, \Gamma|s, o) || P_M(o, r, \Gamma|s) \Big] \Big] && (33) \\
&= \mathbb{E}_{s,\mathcal{T}} \Big[D_{KL} \Big[P_{\mathcal{D}}(o|s, \mathcal{T}) || P_M(o|s) \Big] \Big] + \mathbb{E}_{o,s,\mathcal{T}} \Big[D_{KL} \Big[Q(r, \Gamma|s, o) || P_M(r, \Gamma|s, o) \Big] \Big] && (34) \\
&\geq \mathcal{J}. && (35)
\end{aligned}$$

Where the inequality follows from the the positivity of the Kullback-Leibler divergence, i.e. the fact that $\mathbb{E}_{o,s,\mathcal{T}} \left(D_{KL}[Q(r, \Gamma|s, o)||P_M(r, \Gamma|s, o)] \right) \geq 0$. To understand this upper bound, note that we are now effectively performing two optimizations: approximating the posterior over network states (r, Γ), and subsequently approximating the input-output probability distribution. In the event that Q corresponds to the true posterior, the bound is tight, i.e. $\mathcal{J} = \mathcal{L}$. This suggests a coordinate descent procedure called *variational expectation maximization*, where one first minimizes $\mathcal{L}$ with respect to Q (restricted to some convenient class of probability distributions), and then one subsequently minimizes $\mathcal{L}$ with respect to network parameters W. To make this process more concrete, we will first simplify our loss function $\mathcal{L}$:

$$\begin{aligned}
\mathcal{L} &= \mathbb{E}_{s,\mathcal{T}} \Big[D_{KL} \Big[P_{\mathcal{D}}(o|s, \mathcal{T}) Q(r, \Gamma|s, o) || P_M(o, r, \Gamma|s) \Big] \Big] && (36) \\
&= \mathbb{E}_{s,\mathcal{T}} \left[\iiint P_{\mathcal{D}}(o|s, \mathcal{T}) Q(r, \Gamma|s, o) \log \left(\frac{P_{\mathcal{D}}(o|s) Q(r, \Gamma|s, o)}{P_M(o, r, \Gamma|s)} \right) dr do d\Gamma \right] && (37) \\
&\equiv \mathbb{E}_{s,o,\mathcal{T},Q} \Big[\log Q(r, \Gamma|s, o) - \log P_M(o, r, \Gamma|s) \Big] && (38) \\
&= \mathbb{E}_{s,o,\mathcal{T},Q} \Big[-\log P_M(o, r, \Gamma|s) \Big] + \mathbb{E}_{s,o,\mathcal{T},Q} \Big[H(Q(r, \Gamma|s, o)) \Big], && (39)
\end{aligned}$$

where we use $\mathbb{E}_Q$ to indicate expectations over r and Γ variables sampled from the approximate posterior, and $H(Q(r, \Gamma|s, o))$ is the entropy of Q. We will choose a maximum *a posteriori* (MAP)

point estimate for our Q distribution over r and Γ, where we assume Γ is an arbitrary constant for each task $\mathcal{T}$ and r is allowed to vary with stimuli s. Because point estimates have constant entropy, we can ignore the entropy term for the purposes of optimization, i.e.:

$$\mathcal{L} \equiv \mathbb{E}_{s,o,\mathcal{T}}\Big[-\log P_M\left(o, r^*, \Gamma^* | s\right)\Big], \tag{40}$$

where r^* is the hidden state estimate for stimulus s, and Γ^* is the gain estimate for all stimuli sampled from the support set for a given training epoch; we describe how these estimates are calculated in more detail below. The resultant loss is simply the negative log-likelihood of P_M, evaluated at inferred hidden network states r^* and Γ^*. This leads to the following expanded loss, substituting in the conditional $\log$-probability distributions for the hidden and output network layers, along with the sparse Laplace prior term for Γ gating variables:

$$\mathcal{L} = \mathbb{E}_{s,o,\mathcal{T}}[-\log P_M(r^* \mid s, \Gamma_1^*) - \log P_M(o \mid r^*, \Gamma_2^*) - \log P(\Gamma_1^*, \Gamma_2^*)] \tag{41}$$

$$= \mathbb{E}_{s,o,\mathcal{T}}\left[\frac{1}{2\sigma_1^2}\left\|r^* - \overline{W}_1(\Gamma_1^*)s\right\|_2^2 + \frac{1}{2\sigma_2^2}\left\|o - \overline{W}_2(\Gamma_2^*)r^*\right\|_2^2 + \alpha_1\|\Gamma_1^*\|_1 + \alpha_2\|\Gamma_2^*\|_1\right], \tag{42}$$

where σ_1^2, σ_2^2, α_1, and α_2 are model hyperparameters. This loss has to be minimized for r^*, Γ^*, and W. There are many ways to carry out this optimization procedure in principle; in what follows, we will detail one particular method with convenient properties, based on coordinate descent. We observe that for network states r^* this optimization reduces to a simple, local regression problem if W and Γ^* are held fixed—the same holds true for W and Γ^*. Exploiting this fact, we employ the following coordinate descent scheme (detailed in Algorithm 1). For each epoch of training, network states r^* and multiplicative gain parameters Γ are inferred from a small support set. Subsequently, we hold Γ^* fixed, and optimize for network states r^* and parameters W on the remainder of the data. Interestingly, this procedure does not require explicit knowledge of whether the task $\mathcal{T}$ has changed for the given epoch: optimal Γ^* variables simply change when dictated by changes in the statistics of (s, o) pairs, relying only on the assumption that the task remains the same for the duration of each epoch.

Below, we explain exactly how optimization occurs for r^*, Γ^*, and W. Weight updates are closely related to difference target propagation (Bengio, 2014; Lee et al., 2015), where layer is assigned a local target state r^* and trained toward it with a local, per-layer loss instead of a single backpropagated global gradient; we adapt this by deriving each layer's target via Kalman-like update rather than the autoencoder-based inverse mappings of classical difference target propagation (Lee et al., 2015). At test time, the shared weights are frozen and only the gamma parameters are inferred again, to assess the model's ability to generalize to new data.

Unlike standard backpropagation-based learning, GaTaP gives the model two distinct adaptation mechanisms. The gamma parameters provide task-specific routing, while the shared subnetworks store reusable knowledge. Therefore, if two tasks share useful structure, they may activate similar gamma patterns even if they are not identical tasks.

A.1.2 Derivation of the target using the Kalman update in the linear case

In this section, we provide the methodology for inferring the MAP estimate r^*, conditioned on W, Γ and the input-output pair (s, o). While methods based on predictive coding (Rao & Ballard, 1999; Whittington & Bogacz, 2017) would perform gradient descent on Equation 40 to calculate r^*, we will instead analytically compute the posterior distribution for r^* as well as its mode. In the linear Gaussian case, computing this posterior is simple; in the *non*linear Gaussian case, we can approximate the posterior via a Taylor expansion, as detailed in the next section (Appendix A.2).

Recall: Conditional Gaussian Distribution

For a joint Gaussian distribution,

$$\begin{bmatrix} x \\ y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{bmatrix}\right), \tag{43}$$

the conditional distribution is given by:

$$x|y \sim \mathcal{N}\left(\mu_x + \Sigma_{xy}\Sigma_{yy}^{-1}(y - \mu_y), \Sigma_{xx} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{yx}\right). \tag{44}$$

Kalman update

As in the previous section, we assume the following:

$$P_M(r \mid s, \Gamma_1) = \mathcal{N}\left(\overline{W}_1(\Gamma_1)\, s, \Sigma_1\right) \qquad \text{and} \qquad P_M(o \mid r, \Gamma_2) = \mathcal{N}\left(\overline{W}_2(\Gamma_2)\, r, \Sigma_2\right), \quad (45)$$

Where $\Sigma_1 = \epsilon_1 I$ and $\Sigma_2 = \epsilon_2 I$. Thus, the model defines the joint distribution

$$P_M(r, o \mid s, \Gamma) = P_M(o \mid r, \Gamma_2) P_M(r \mid s, \Gamma_1). \quad (46)$$

Because both distributions are Gaussian and the mapping from r to o is linear, the conditional joint distribution of r and o given s is also Gaussian:

$$P_M\left(\begin{bmatrix} r \\ o \end{bmatrix} \middle| s, \Gamma\right) = \mathcal{N}\left(\begin{bmatrix} \overline{W}_1(\Gamma_1)\, s \\ \overline{W}_2(\Gamma_2)\, \overline{W}_1(\Gamma_1)\, s \end{bmatrix}, \begin{bmatrix} \Sigma_1 & \Sigma_1 \overline{W}_2(\Gamma_2)^T \\ \overline{W}_2(\Gamma_2)\, \Sigma_1 & \overline{W}_2(\Gamma_2)\, \Sigma_1 \overline{W}_2(\Gamma_2)^T + \Sigma_2 \end{bmatrix}\right). \quad (47)$$

Using this joint distribution and Equation 44, we can infer the mean of the hidden state conditioned on the observed stimulus and output as well as the fixed Γ variables:

$$\mathbb{E}_{p_M(r \mid o, s, \Gamma)}[r] = \overline{W}_1(\Gamma_1)\, s + \Sigma_1 \overline{W}_2(\Gamma_2)^T \left(\overline{W}_2(\Gamma_2)\, \Sigma_1 \overline{W}_2(\Gamma_2)^T + \Sigma_2\right)^{-1} \left(o - \overline{W}_2(\Gamma_2)\, \overline{W}_1(\Gamma_1)\, s\right). \quad (48)$$

We define the Kalman gain

$$K = \Sigma_1 \overline{W}_2(\Gamma_2)^T \left(\overline{W}_2(\Gamma_2)\, \Sigma_1 \overline{W}_2(\Gamma_2)^T + \Sigma_2\right)^{-1}. \quad (49)$$

Then the posterior mean can be written more compactly as

$$\mathbb{E}_{p_M(r \mid o, s)}[r] = \overline{W}_1(\Gamma_1)\, s + K(o - \hat{o}), \quad (50)$$

where

$$\hat{o} = \overline{W}_2(\Gamma_2)\, \overline{W}_1(\Gamma_1)\, s \quad (51)$$

is the feedforward prediction of the output. For a Gaussian posterior, the mean and the mode are the same. Therefore, the posterior mean gives the optimal inferred hidden state under the linear Gaussian model. We denote this inferred state by

$$r^\star = \bar{r} + K(o - \hat{o}), \quad (52)$$

where $\bar{r} = \overline{W}_1(\Gamma_1)\, s$. This expression shows that the inferred hidden target is the mean unconditioned feedforward hidden state plus an additive correction term. The correction depends on the output error

$$e_o = o - \hat{o},$$

and the Kalman gain K determines how much of this output error should be assigned to the hidden state.

This derivation provides the basic target propagation rule in the linear case. The weights are optimized to minimize the mean squared error between the the actual hidden state provided by the model and the target. The target hidden state is not obtained by backpropagating gradients through the network. Instead, it is inferred probabilistically as the hidden state that best 'explains' both the input and the observed output under the model.

A.2 Nonlinear one hidden layer neural network: Extended Kalman update

The previous derivation relied on a linear relationship between the hidden state and the output. In this section, we show how the same Kalman-style target update can be extended to a nonlinear one-hidden-layer network by using a local linear approximation, similar to the extended Kalman filter (Ribeiro, 2004).

We consider the model

$$r \mid s \sim \mathcal{N}\left(\overline{W}_1(\Gamma_1)\, s, \Sigma_1\right), \quad (53)$$

$$o \mid r \sim \mathcal{N}\left(\overline{W}_2(\Gamma_2)\, f(r), \Sigma_2\right), \quad (54)$$

where s is the input, r is the hidden state, o is the output, f is a nonlinear activation function, and $\overline{W}_2$ maps the activated hidden state to the output.

Because $f(r)$ is nonlinear, the posterior distribution $p_M(r \mid o, s)$ is not Gaussian in general. Therefore, we approximate the nonlinear mapping locally. Let r_0 be the current feedforward hidden state, and let

$$J_f = \left.\frac{\partial f}{\partial r}\right|_{r=r_0} \tag{55}$$

be the Jacobian of the activation function evaluated at r_0. Using a first-order Taylor expansion,

$$f(r) \approx f(r_0) + J_f(r - r_0). \tag{56}$$

Substituting this approximation into the output model gives

$$o \approx \overline{W}_2\left(\Gamma_2\right) f(r_0) + \overline{W}_2\left(\Gamma_2\right) J_f(r - r_0) + \epsilon_2, \qquad \epsilon_2 \sim \mathcal{N}(0, \Sigma_2). \tag{57}$$

For simplicity, define the local linear map

$$H = \overline{W}_2\left(\Gamma_2\right) J_f. \tag{58}$$

Then the linearized output model becomes

$$o \approx \overline{W}_2\left(\Gamma_2\right) f(r_0) + H(r - r_0) + \epsilon_2. \tag{59}$$

Under this local linear approximation, the joint distribution of r and o conditioned on s is approximately Gaussian:

$$p_M\left(\begin{bmatrix} r \\ o \end{bmatrix} \middle| s\right) \approx \mathcal{N}\left(\begin{bmatrix} r_0 \\ \overline{W}_2\left(\Gamma_2\right) f(r_0) \end{bmatrix}, \begin{bmatrix} \Sigma_1 & \Sigma_1 H^T \\ H\Sigma_1 & H\Sigma_1 H^T + \Sigma_2 \end{bmatrix}\right). \tag{60}$$

Therefore, the mean of the approximate posterior distribution $p_M(r \mid o, s)$ is

$$\mathbb{E}_{p_M(r|o,s)}[r] \approx r_0 + \Sigma_1 H^T \left(H\Sigma_1 H^T + \Sigma_2\right)^{-1} \left(o - \overline{W}_2\left(\Gamma_2\right) f(r_0)\right). \tag{61}$$

Replacing H by $\overline{W}_2\left(\Gamma_2\right) J_f$, we obtain the nonlinear target update

$$r^\star = r_0 + \Sigma_1 J_f^T \overline{W}_2\left(\Gamma_2\right)^T \left(\overline{W}_2\left(\Gamma_2\right) J_f \Sigma_1 J_f^T \overline{W}_2\left(\Gamma_2\right)^T + \Sigma_2\right)^{-1} \left(o - \overline{W}_2\left(\Gamma_2\right) f(r_0)\right). \tag{62}$$

Equivalently, defining the Kalman gain

$$K = \Sigma_1 J_f^T \overline{W}_2\left(\Gamma_2\right)^T \left(\overline{W}_2\left(\Gamma_2\right) J_f \Sigma_1 J_f^T \overline{W}_2\left(\Gamma_2\right)^T + \Sigma_2\right)^{-1}, \tag{63}$$

the hidden target can be written as

$$r^\star = r_0 + K\left(o - \hat{o}\right), \qquad \hat{o} = \overline{W}_2\left(\Gamma_2\right) f(r_0). \tag{64}$$

Thus, as in the linear case, the inferred target is the current feedforward hidden state plus an error-correction term. The main difference is that the linear weight matrix $\overline{W}_2$ is replaced by the local linearization $\overline{W}_2 J_f$. This gives an analytic approximation to the optimal hidden state, instead of requiring iterative gradient descent to infer the target. Specifically, this single-step closed-form update mirrors the Gauss-Newton formulation of target propagation (Meulemans et al., 2020), where the regularized pseudo-inverse of the local Jacobian is used to analytically jump to the target rather than iteratively optimizing toward it.

A.3 Target propagation in deep networks with multiplicative gains

The previous section showed how to infer a target state for a nonlinear network with one hidden layer. We now extend the same idea to a deep network, where a target state must be inferred at every layer. Let $h_0 \in \mathbb{R}^{d_0}$ denote the input and let $h_i \in \mathbb{R}^{d_i}$ denote the hidden state at layer i. For a feedforward network with L layers, we define the gated hidden state as

$$h_i = \sum_{k=1}^{n_i} \Gamma_i^{(k)} \odot \left(W_i^{(k)} \phi_{i-1}(h_{i-1})\right) + \epsilon_i, \qquad \text{with } \epsilon_i \sim \mathcal{N}\left(0, \Sigma_i\right), \qquad i = 1, \ldots, L. \tag{65}$$

Here, n_i is the number of subnetworks connecting layer $i-1$ to layer i, $W_i^{(k)} \in \mathbb{R}^{d_i \times d_{i-1}}$ is the k-th subnetwork weight matrix, ϕ_{i-1} is the activation function applied to the previous hidden state and ϕ_0 is just the identity function. The vector $\Gamma_i^{(k)} \in \mathbb{R}^{d_i}$ gates the rows of $W_i^{(k)}$. Equivalently, we can define an effective task-dependent weight matrix

$$\overline{W}_i = \sum_{k=1}^{n_i} \Gamma_i^{(k)} \odot W_i^{(k)}, \tag{66}$$

so that

$$h_i = \overline{W}_i \phi_{i-1}(h_{i-1}) + \epsilon_i, \qquad \text{with } \epsilon_i \sim \mathcal{N}(0, \Sigma_i), \qquad i = 1, \ldots, L. \tag{67}$$

The weights $W_i^{(k)}$ are shared across tasks, while the gain parameters are inferred separately in each training epoch (agnostic to whether or not the task $\mathcal{T}$ has changed). In the next sections, we describe how to compute MAP estimates for the gains Γ^* and the network hidden states h_i^*.

A.3.1 Close-form solution for the $\Gamma_i^{(k)}$.

The closed-form update for Γ enables fast and sparse task adaptation without an additional gradient-based inner optimization. Inferring $\Gamma^\star$ before updating the shared weights first determines the task-specific computational route, after which the outer optimization modifies the corresponding shared components while leaving less relevant pathways unaffected. This separation supports fast task inference while reducing interference during slower shared-weight learning.

Let consider the equation: $h_i = \sum_{k=1}^{n_i} \Gamma_i^{(k)} \odot \left(W_i^{(k)} \phi_{i-1}(h_{i-1}) \right) + \epsilon_i,\ i = 2, \ldots, L$. Here, $\Gamma_i^{(k)} = \left[\gamma_{i,1}^{(k)}, \ldots, \gamma_{i,d_i}^{(k)} \right], h_i \in \mathbb{R}^{d_i}$ and $W_i^{(k)} \in \mathbb{R}^{d_i \times d_{i-1}}$. Let $a_{i-1,p} = \phi_{i-1}(h_{i-1,p})$ for a data point p in a batch of P data points. For subnetwork k, define $Z_{i,p}^{(k)} = W_i^{(k)} \phi_{i-1}(h_{i-1,p})$ so that $h_{i,p} = \sum_{k=1}^{n_i} \Gamma_i^{(k)} \odot Z_{i,p}^{(k)}$. Let assume for layer i we have the target $h_{i,p}^*$ (we will discuss how these are obtained in the next section, using Equation 84). The local loss (on the support set) is defined as follows:

$$\mathcal{L}_{i,\text{support}} \left(\Gamma_i^{(k)} \right) = \frac{1}{2P} \sum_{p=1}^{P} \left\| h_{i,p}^* - h_{i,p} \right\|_2^2 + \alpha_i \sum_{k=1}^{n_i} \left\| \Gamma_i^{(k)} \right\|_1 \tag{68}$$

$$= \frac{1}{2P} \sum_{p=1}^{P} \left\| h_{i,p}^* - \sum_{k=1}^{n_i} \Gamma_i^{(k)} \odot Z_{i,p}^{(k)} - \epsilon_i \right\|_2^2 + \alpha_i \sum_{k=1}^{n_i} \left\| \Gamma_i^{(k)} \right\|_1 \tag{69}$$

$$= \frac{1}{2P} \sum_{p=1}^{P} \sum_{q=1}^{d_i} \left(h_{i,p,q}^* - \sum_{k=1}^{n_i} \gamma_{i,q}^{(k)} Z_{i,p,q}^{(k)} - \epsilon_i \right)^2 + \alpha_i \sum_{k=1}^{n_i} \sum_{q=1}^{d_i} \left| \gamma_{i,q}^{(k)} \right|. \tag{70}$$

For a fixed output hidden neuron q, we have:

$$\mathcal{L}_{i,k,q} \left(\gamma_{i,q}^{(k)} \right) = \frac{1}{2P} \sum_{p=1}^{P} \left(R_{i,p,q}^{-k} - \gamma_{i,q}^{(k)} Z_{i,p,q}^{(k)} \right)^2 + \alpha_i \left| \gamma_{i,q}^{(k)} \right| + \sum_{l \neq k}^{n_i} \left| \gamma_{i,q}^{(l)} \right|, \tag{71}$$

with

$$\mathcal{L}_i \left(\Gamma_i^{(k)} \right) = \sum_{q=1}^{d_i} \mathcal{L}_{i,k,q} \left(\gamma_i^{(k)} \right) \text{ and } R_{i,p}^{-k} = h_{i,p}^* - \sum_{l \neq k}^{d_i} \Gamma_i^{(l)} \odot Z_{i,p}^{(l)} - \epsilon_i. \tag{72}$$

The optimum of $\mathcal{L}_{i,k,q} \left(\gamma_{i,q}^{(k)} \right)$ should either be zero, or $\gamma_{i,q}^{(k)}$ should be zero. We first assume that $\gamma_{i,q}^{(k)}$ is nonzero. Then the derivative of $\mathcal{L}_{i,k,q} \left(\gamma_{i,q}^{(k)} \right)$ with respect to $\gamma_{i,q}^{(k)}$ is given by

$$\nabla_{\gamma_{i,q}^{(k)}} \mathcal{L}_{i,k,q} \left(\gamma_{i,q}^{(k)} \right) = -\frac{1}{P} \sum_{p=1}^{P} Z_{i,p,q}^{(k)} \left(R_{i,p,q}^{-k} - \gamma_{i,q}^{(k)} Z_{i,p,q}^{(k)} \right) + \alpha_i \times \text{sign} \left(\gamma_{i,q}^{(k)} \right) = 0 \tag{73}$$

$$\implies \gamma_{i,q}^{(k)} = \frac{\frac{1}{P} \sum_{p=1}^{P} Z_{i,p,q}^{(k)} R_{i,p,q}^{-k} - \alpha_i \times \text{sign} \left(\gamma_{i,q}^{(k)} \right)}{\frac{1}{P} \sum_{p=1}^{P} \left(Z_{i,p,q}^{(k)} \right)^2}. \tag{74}$$

From this equality, we can infer the following three possible conditions:

$$\text{If } \gamma_{i,q}^{(k)} > 0, \frac{1}{P}\sum_{p=1}^{P} Z_{i,p,q}^{(k)} R_{i,p,q}^{-k} - \alpha_i > 0 \implies \frac{1}{P}\sum_{p=1}^{P} Z_{i,p,q}^{(k)} R_{i,p,q}^{-k} > \alpha_i, \tag{75}$$

$$\text{else if } \gamma_{i,q}^{(k)} < 0, \frac{1}{P}\sum_{p=1}^{P} Z_{i,p,q}^{(k)} R_{i,p,q}^{-k} + \alpha_i < 0 \implies \frac{1}{P}\sum_{p=1}^{P} Z_{i,p,q}^{(k)} R_{i,p,q}^{-k} < -\alpha_i \tag{76}$$

$$\text{else } \gamma_{i,q}^{(k)} = 0. \tag{77}$$

Therefore, the close-form solution for $\gamma_{i,q}^{(k)}$ is given by:

$$\left(\gamma_{i,q}^{(k)}\right)^* = \text{sign}\left(b_{i,k,q}\right) \times \frac{\left\lfloor |b_{i,k,q}| - \alpha_i \right\rfloor^+}{a_{i,k,q}} \tag{78}$$

where

$$a_{i,k,q} = \frac{1}{P}\sum_{p=1}^{P}\left(Z_{i,p,q}^{(k)}\right)^2 \text{ and } b_{i,k,q} = \frac{1}{P}\sum_{p=1}^{P} Z_{i,p,q}^{(k)} R_{i,p,q}^{-k}. \tag{79}$$

Therefore

$$\left(\Gamma_i^{(k)}\right)^* = \left[\left(\gamma_{i,1}^{(k)}\right)^*, \ldots, \left(\gamma_{i,d_i}^{(k)}\right)^*\right]. \tag{80}$$

In principle, calculating optimal h_i^* and $\left(\Gamma_i^{(k)}\right)^*$ values should occur via a coordinate descent procedure, alternating repeatedly between computing the target states and gain variables across layers. In practice, however, we found that only a single iteration was needed for good performance, and so we took this approach to generate our empirical results for the sake of computational efficiency.

A.3.2 Outer-loop optimization

After gamma has been inferred from the support set, the shared weights are updated on the query set. The output target is defined as

$$h_L^\star = h_L + K_L\left(y - h_L\right), \tag{81}$$

where y is the target label representation and K_L controls the size of the output correction.

For hidden layers, the target is propagated backward through the network. Let

$$F_{i+1}(h_i) = \overline{W}_{i+1}\phi_i(h_i) \tag{82}$$

be the forward map from layer i to layer $i+1$. Its Jacobian is

$$J_i = \frac{\partial F_{i+1}(h_i)}{\partial h_i}. \tag{83}$$

The hidden target is computed as

$$h_i^\star = h_i + K_i\left(h_{i+1}^\star - h_{i+1}\right), \qquad i = L-1, \ldots, 1, \tag{84}$$

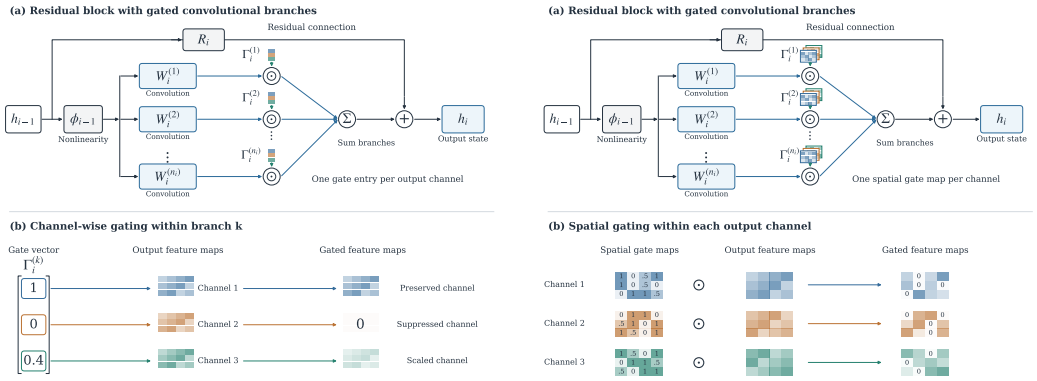
where

$$K_i = \Sigma_i J_i^T\left(J_i \Sigma_i J_i^T + \Sigma_{i+1}\right)^{-1}. \tag{85}$$

Here, $\Sigma_i = \sigma_i^2 I$ controls the uncertainty allowed at layer i.

Once the targets $h_i^\star$ and gains $\Gamma^\star$ are fixed, the update of W_i depends only on the local query loss at layer i. The support objective, sparsity penalty on Γ, and losses from other layers are constant with respect to W_i, since gradients are not propagated through the inferred targets or gains. Hence, for the weight update,

$$\nabla_{W_i}\mathcal{L}_{\text{total}} = \nabla_{W_i}\mathcal{L}_{i,\text{query}}, \tag{86}$$



(a) Channel-wise gating. (b) Spatial gating within each channel.

Figure S1: **Two gating mechanisms for residual convolutional networks.** Both variants combine parallel convolutional branches with task-specific gates and add the residual contribution $R_i(h_{i-1})$ to the sum of the gated branch outputs. **Left:** Channel-wise gating assigns one scalar to each output channel, uniformly scaling its entire feature map. **Right:** Spatial gating assigns a separate gate value to each spatial location within each output channel, enabling selective modulation within a feature map. The lower panels illustrate how gate values preserve, suppress, or attenuate the corresponding activations.

up to any explicit regularization applied directly to W_i. Given these targets, the local weight-learning objective at layer i is

$$\mathcal{L}_{i,\text{query}} = \frac{1}{2P'} \sum_{p=1}^{P'} \left\| h^{\star}_{i,p} - \sum_{k=1}^{n_i} \left(\Gamma_i^{(k)}\right)^{\star} W_i^{(k)} \phi_{i-1}(h_{i-1,p}) - \epsilon_i \right\|^2_{\Sigma_i^{-1}}, \tag{87}$$

where P' is the number of query examples. During this update, the gamma parameters are kept fixed. Only the shared weights $W_i^{(k)}$ are optimized.

The training procedure can therefore be summarized as

$$\Gamma_T^{\star} = \arg\min_{\Gamma} \mathcal{L}_{\text{support}}\left(\Gamma; W, T\right) + \sum_{i,k} \alpha_i \left\| \gamma_i^{(k)} \right\|_1, \tag{88}$$

followed by

$$W \leftarrow W - \eta \nabla_W \mathcal{L}_{\text{query}}\left(W, \Gamma_T^{\star}; T\right). \tag{89}$$

This is the modified inner-loop and outer-loop optimization used in the proposed framework. The support set is used only for gamma inference, and the query set is used only for updating the shared weights.

At test time, the update of W is removed. The weights are frozen, gamma is inferred from the support set, and accuracy and loss are computed on the query set. The full algorithm for the training procedure is given in [1].

A.4 Convolutional Neural Networks

CNNs are needed to evaluate GaTaP on realistic image tasks, where spatial structure and local feature extraction are essential. Extending the framework to convolutions tests whether the same gating and target-inference mechanisms remain effective beyond fully connected networks. Gating in a convolutional layer could be accomplished by assigning one gate per output channel, applying the same gate to *all spatial locations*, or a separate gate could be assigned for each channel at each spatial position (see Supplemental Figure S1b for a schematic). We begin with the "spatial gating" version, before introducing the "channel gating" version. We find empirically that spatial gating is more effective (Supplemental Figure S4), and so we use this gating mechanism for our main results.

A.4.1 CASE OF MULTI SUBNETWORKS: SPATIAL GATING

For a convolutional layer with multiple subnetworks, we would have kernels

$$W_i^{(k)} \in \mathbb{R}^{C_i \times I_i \times k_{h_i} \times k_{w_i}}, \tag{90}$$

where C_i is the feature output number of channel, I_i is the input number of channel, k_{h_i}, k_{w_i} are the height and width of the kernels.

We propose the following model for the convolutional layer:

$$h_i = R_i\left(h_{i-1}\right) + \sum_{k=1}^{n_i} \Gamma_i^{(k)} \odot \left(W_i^{(k)} * \phi_{i-1}(h_{i-1})\right) + \epsilon_i, \quad \text{for } i = 2, \ldots, L, \tag{91}$$

where

- n_i : number of kernels or shared knowledge between tasks,
- ϕ_{i-1} : is the nonlinearity function,
- $\Gamma_i^{(k)} \in \mathbb{R}^{C_i}$ plays the role of gating mechanism across the rows of $W_i^{(k)} \in \mathbb{R}^{C_i \times I_i \times k_{h_i} \times k_{w_i}}$.

Let $a_{i-1,p} = \phi_{i-1}(h_{i-1,p})$ for an example p in the support set $p = 1, \ldots, P$. For subnet k, let $Z_{i,p}^{(k)} = W_i^{(k)} * a_{i-1,p}$. Assume a convolution layer has C_i output channels, then $Z_{i,p}^{(k)} \in \mathbb{R}^{C_i \times H_i \times W_i}$. Let $u = (u_1, u_2)$ be the spatial position in the output feature map. So $Z_{i,p,c,u}^{(k)}$ is such that p is the example index, c is the output channel, u is the spatial position and k is the subnetwork index. $\Gamma_i^{(k)} \in \mathbb{R}^{C_i \times H_i \times W_i}$ so that $\left[\Gamma_i^{(k)} Z_{i,p}^{(k)}\right]_{c,u} = \gamma_{i,c,u}^{(k)} Z_{i,p,c,u}^{(k)}$ and

$$h_{i,p,c,u} = \sum_{k=1}^{n_i} \gamma_{i,c,u}^{(k)} Z_{i,p,c,u}^{(k)}. \tag{92}$$

Let assume we have $h_{i,p,c,u}^*$ which is the target hidden state. We define the local loss as:

$$\mathcal{L}_i\left(\Gamma_i^{(k)}\right) = \frac{1}{2P|\Omega_i|} \sum_{p=1}^{P} \sum_{c=1}^{C_i} \sum_{u \in \Omega_i} \left(h_{i,p,c,u}^* - R_i\left(h_{i-1}\right) - \sum_{k=1}^{n_i} \gamma_{i,c,u}^{(k)} Z_{i,p,c,u}^{(k)} - \epsilon_i\right)^2 + \alpha_i \sum_{k=1}^{n_i} \sum_{c=1}^{C_i} \sum_{u \in \Omega_i} \left|\gamma_{i,c,u}^{(k)}\right|. \tag{93}$$

Let

$$R_{i,p,c,u}^{(-k)} = h_{i,p,c,u}^* - R_i\left(h_{i-1}\right) - \sum_{l \neq k} \gamma_{i,c,u}^{(l)} Z_{i,p,c,u}^{(l)} - \epsilon_i. \tag{94}$$

For $\gamma_{i,c,u}^{(k)}$, we have its corresponding loss given by:

$$\mathcal{L}_{i,k,c,u}\left(\gamma_{i,c}^{(k)}\right) = \frac{1}{2P|\Omega_i|} \sum_{p=1}^{P} \left(R_{i,p,c,u}^{(-k)} - \gamma_{i,c,u}^{(k)} Z_{i,p,c,u}^{(k)}\right)^2 + \alpha_i \left|\gamma_{i,c,u}^{(k)}\right|. \tag{95}$$

With the following definitions:

$$b_{i,k,c,u} = \frac{1}{P|\Omega_i|} \sum_{p=1}^{P} Z_{i,p,c,u}^{(k)} R_{i,p,c,u}^{(-k)} \quad \text{and} \quad a_{i,k,c,u} = \frac{1}{P|\Omega_i|} \sum_{p=1}^{P} \left(Z_{i,p,c,u}^{(k)}\right)^2, \tag{96}$$

the closed-form solution for $\gamma_{i,c,u}^{(k)}$ (directly following from Equation 78) is given by:

$$\left(\gamma_{i,c,u}^{(k)}\right)^* = \operatorname{sign}\left(b_{i,k,c,u}\right) \times \left(\frac{\left\lfloor |b_{i,k,c,u}| - \alpha_i \right\rfloor^+}{a_{i,k,c,u}}\right). \tag{97}$$

A.4.2 CASE OF MULTI SUBNETWORKS: CHANNEL GATING

Let $a_{i-1,p} = \phi_{i-1}(h_{i-1,p})$ for an example p in the support set $p = 1, \ldots, P$. For subnet k, let $Z_{i,p}^{(k)} = W_i^{(k)} * a_{i-1,p}$. Assume a convolution layer has C_i output channels, then $Z_{i,p}^{(k)} \in \mathbb{R}^{C_i \times H_i \times W_i}$. So $Z_{i,p,c}^{(k)}$ is such that p is the example index, c is the output channel, and k is the subnetwork index. $\Gamma_i^{(k)} \in \mathbb{R}^{C_i}$ so that $\left[\Gamma_i^{(k)} Z_{i,p}^{(k)}\right]_c = \gamma_{i,c}^{(k)} Z_{i,p,c}^{(k)}$ and

$$h_{i,p,c} = \sum_{k=1}^{n_i} \gamma_{i,c}^{(k)} Z_{i,p,c}^{(k)}. \tag{98}$$

Let assume we have $h_{i,p,c}^*$ which is the target hidden state. We define the local loss as:

$$\mathcal{L}_i\left(\Gamma_i^{(k)}\right) = \frac{1}{2P|\Omega_i|} \sum_{p=1}^{P} \sum_{c=1}^{C_i} \left(h_{i,p,c}^* - R_i\left(h_{i-1}\right) - \sum_{k=1}^{n_i} \gamma_{i,c}^{(k)} Z_{i,p,c}^{(k)} - \epsilon_i \right)^2 + \alpha_i \sum_{k=1}^{n_i} \sum_{c=1}^{C_i} \left|\gamma_{i,c}^{(k)}\right|. \tag{99}$$

Let

$$R_{i,p,c}^{(-k)} = h_{i,p,c}^* - R_i\left(h_{i-1}\right) - \sum_{l \neq k} \gamma_{i,c}^{(l)} Z_{i,p,c}^{(l)} - \epsilon_i. \tag{100}$$

For $\gamma_{i,c}^{(k)}$, we have its corresponding loss given by:

$$\mathcal{L}_{i,k,c}\left(\gamma_{i,c}^{(k)}\right) = \frac{1}{2P|\Omega_i|} \sum_{p=1}^{P} \left(R_{i,p,c}^{(-k)} - \gamma_{i,c}^{(k)} Z_{i,p,c}^{(k)} \right)^2 + \alpha_i \left|\gamma_{i,c}^{(k)}\right|.$$

With the following definitions:

$$b_{i,k,c} = \frac{1}{P|\Omega_i|} \sum_{p=1}^{P} Z_{i,p,c}^{(k)} R_{i,p,c}^{(-k)} \quad \text{and} \quad a_{i,k,c} = \frac{1}{P|\Omega_i|} \sum_{p=1}^{P} \left(Z_{i,p,c}^{(k)}\right)^2,$$

the closed-form solution for $\gamma_{i,c}^{(k)}$ (again following from Equation 78) is given by

$$\left(\gamma_{i,c}^{(k)}\right)^* = \text{sign}\left(b_{i,k,c}\right) \times \left(\frac{\lfloor |b_{i,k,c}| - \alpha_i \rfloor^+}{a_{i,k,c}} \right).$$

A.4.3 TARGET HIDDEN STATES

Let

$$h_i = F_i\left(h_{i-1}\right) = R_i\left(h_{i-1}\right) + \sum_{k=1}^{n_i} \Gamma_i^{(k)} \odot \left(W_i^{(k)} * \phi_i\left(h_{i-1}\right) \right) + \epsilon_i$$

The same target update rule, as in Eq. 84, is used in the convolutional case. Let $\Sigma_{i-1} = \sigma_{i-1}^2 I$, $\Sigma_i = \sigma_i^2 I$, and $e_i = h_i^* - h_i$. Therefore

$$\begin{aligned} h_{i-1}^* &= h_{i-1} + \sigma_{i-1}^2 J_i^T \left(\sigma_{i-1}^2 J_i J_i^T + \sigma_i^2 I\right)^{-1} e_i \\ &= h_{i-1} + \sigma_{i-1}^2 J_i^T \left(\sigma_{i-1}^2 J_i J_i^T + \sigma_i^2 I\right)^{-1} e_i \end{aligned}$$

For large convolutional networks, the Kalman gain K_i is computationally very expensive because of the large inverse calculation $\left(\sigma_{i-1}^2 J_i J_i^T + \sigma_i^2 I\right)^{-1}$. We therefore use a diagonal approximation in our convolutional experiments that renders inversion trivial.

Use of the diagonal of $J_i J_i^T$: Here, for convolutional neural network models, we simply take a diagonal approximation of the $J_i J_i^T$ matrix in calculating the Kalman gain as follows:

$$\begin{aligned} h_{i-1}^* &= h_{i-1} + \sigma_{i-1}^2 J_i^T \left(\sigma_{i-1}^2 J_i J_i^T + \sigma_i^2 I\right)^{-1} e_i \\ &\approx h_{i-1} + \sigma_{i-1}^2 J_i^T \left(\sigma_{i-1}^2 \text{diag}\left(J_i J_i^T\right) + \sigma_i^2 I\right)^{-1} e_i \\ &= h_{i-1} + \sigma_{i-1}^2 J_i^T \left[\text{diag}\left(\sigma_{i-1}^2 J_i J_i^T + \sigma_i^2 I\right)\right]^{-1} e_i. \end{aligned}$$

Since inverting a diagonal matrix is equivalent to pointwise inversion of each diagonal matrix element, this makes computing the inverse much simpler. In practice, we find that it does not significantly impact training performance.

Algorithm 1 Gated Target Propagation

Training phase
Initialize all shared subnetworks $\{W_i^{(k)}\}$
for $j = 1, \ldots, m$ **do**
  Receive task $\mathcal{T}_j$
  Split task data into support set $\mathcal{S}_j$ and query set $\mathcal{Q}_j$
  for $e = 1, \ldots, E$ **do**
    Inner loop: infer task-specific gates
    Forward $\mathcal{S}_j$ through the network
    Compute the output target $h_L^\star$ followed by $\Gamma_{j,L}^\star$
    for $i = L-1, \ldots, 1$ **do**
      Compute target state $h_i^\star$ using target propagation
      Infer gamma parameters $\Gamma_{j,i}^\star$ from $\mathcal{S}_j$ using the closed-form sparse update
    end for
    Outer loop: update shared weights
    Keep $\Gamma_j^\star$ fixed
    Forward $\mathcal{Q}_j$ through the gated network
    Compute query target states $\{h_i^\star\}_{i=1}^L$
    Compute the local target-propagation loss $\mathcal{L}_{i,\text{query}}$
    Update only the shared weights
    $W_i^{(k)} \leftarrow W_i^{(k)} - \eta \nabla_{W_i^{(k)}} \mathcal{L}_{i,\text{query}}$
  end for
end for
Testing phase
Freeze all shared subnetworks $\{W_i^{(k)}\}$
for each evaluation task $\mathcal{T}$ **do**
  Split task data into support set $\mathcal{S}$ and query set $\mathcal{Q}$
  Infer gamma parameters $\Gamma_{\mathcal{T}}^\star$ from $\mathcal{S}$ only
  Evaluate loss and accuracy on $\mathcal{Q}$
end for

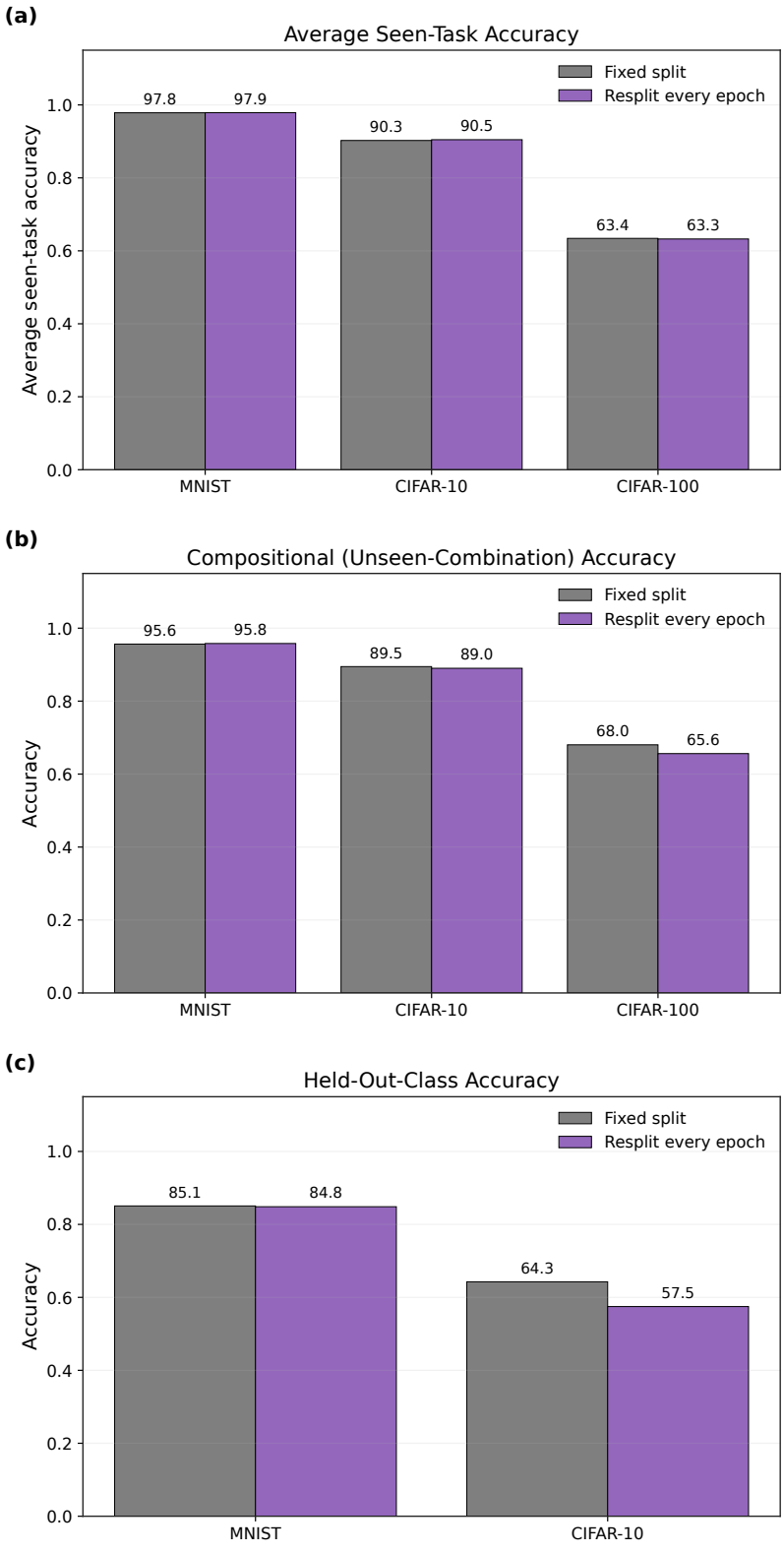


Figure S2: Resplitting the support and query sets every epoch, as compared to a fixed split used across all epochs, as evaluated by **a)** average seen-task accuracy, **b)** compositional (unseen-combination) accuracy, and **c)** held-out-class accuracy.

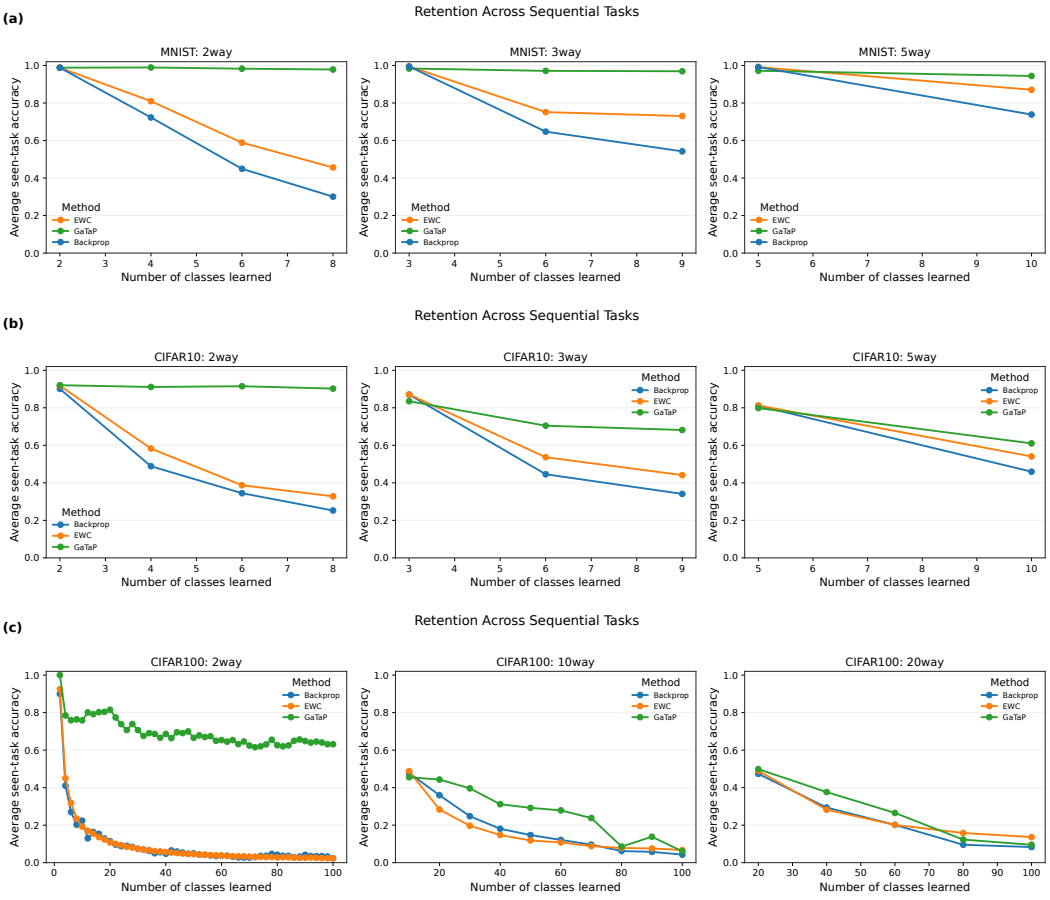


Figure S3: Retention across sequential tasks with varying numbers of classes per-task. Networks are trained with GaTaP (green), EWC (orange), and standard backpropagation (blue). **a)** Models trained on MNIST, for two-way, three-way and five-way continual learning classification tasks. **b)** Models trained on CIFAR-10, for two-way, three-way, and five-way continual learning classification tasks. **c)** Models trained on CIFAR-100, for two-way, ten-way, and twenty-way continual learning classification tasks.

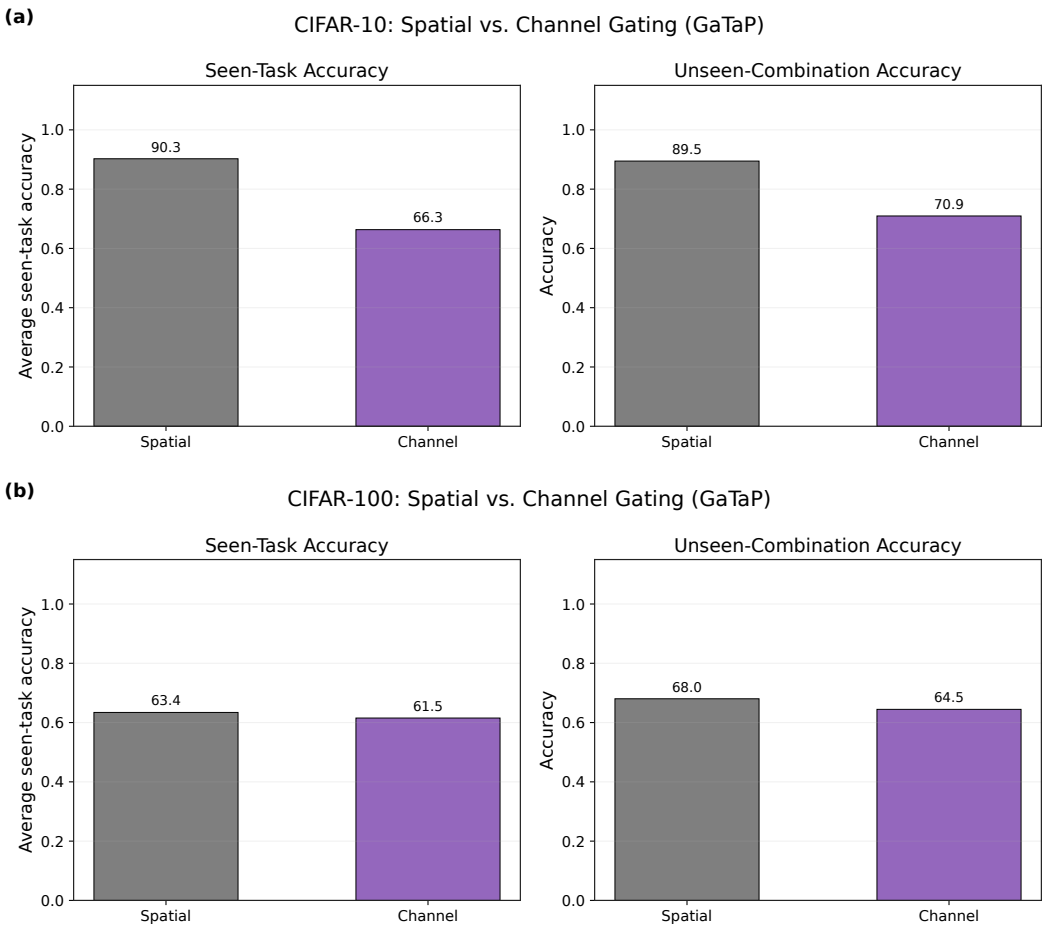


Figure S4: A comparison of seen- (left) and unseen-combination (right) task accuracy when using GaTaP in conjunction with spatial vs. channel gating for convolutional neural network models on **a)** CIFAR-10 and **b)** CIFAR-100.

A.5 HYPERPARAMETERS

Table 1: Shared training hyperparameters for MNIST (2-way, 3-way, and 5-way protocols). Values are identical across all three protocols and all three methods (GaTaP, EWC, Backprop) except where noted.

Hyperparameter	Value
Architecture	Feedforward MLP
Hidden layer sizes	64, 128
Epochs per task	10
Learning rate (lr_w)	1×10^{-3}
Optimizer	Adam
Random seed	42

Table 2: Method-specific hyperparameters for MNIST. GaTaP columns apply identically across the 2-way, 3-way, and 5-way protocols.

GaTaP	
Number of subnetworks	1
Sparsity coefficient (α)	10^{-5}
Gamma inference steps (train / test)	10 / 10
Noise scale by layer (σ)	0.1, 0.1, 0.1
Support examples per class (train / test)	32 / 64
Query batch size (train / test)	128 / 256
EWC	
EWC regularization strength (λ)	20,000
Fisher information sample size	1,000
Fisher information batch size	32

Table 3: Shared training hyperparameters for CIFAR-10 (2-way, 3-way, and 5-way protocols). Values are identical across all three protocols and all three methods (GaTaP, EWC, Backprop) except where noted.

Hyperparameter	Value
Architecture	Simple ResNet (CNN)
Channel widths	16, 32, 64, 128, 256, 512
Kernel size	5
Epochs per task	10
Learning rate (lr_w)	1×10^{-3}
Optimizer	Adam
Random seed	42

Table 4: Method-specific hyperparameters for CIFAR-10. GaTaP columns apply identically across the 2-way, 3-way, and 5-way protocols.

GaTaP	
Number of subnetworks	1
Gamma mode	Spatial
Sparsity coefficient (α)	10^{-5}
Gamma scale	1.0
Gamma inference steps (train / test)	10 / 10
Noise scale by layer (σ)	0.1 (all 7 layers)
Support examples per class (train / test)	20 / 64
Query batch size (train / test)	128 / 256
EWC	
EWC regularization strength (λ)	5,000
Fisher information sample size	1,000
Fisher information batch size	32

Table 5: Shared training hyperparameters for CIFAR-100 (2-way, 10-way, and 20-way protocols). Values are identical across all three protocols and all three methods (GaTaP, EWC, Backprop) except where noted.

Hyperparameter	Value
Architecture	Simple ResNet (CNN)
Channel widths	16, 32, 64, 128, 256, 512
Kernel size	5
Epochs per task	20
Learning rate (lr_w)	3×10^{-4}
Optimizer	Adam
Random seed	42

Table 6: Method-specific hyperparameters for CIFAR-100. GaTaP columns apply identically across the 2-way, 10-way, and 20-way protocols.

GaTaP	
Number of subnetworks	2
Gamma mode	Spatial
Sparsity coefficient (α)	10^{-7}
Gamma scale	1.0
Gamma inference steps (train / test)	10 / 10
Noise scale by layer (σ)	0.1 (all 7 layers)
Support examples per class (train / test)	20 / 64
Query batch size (train / test)	128 / 256
EWC	
EWC regularization strength (λ)	100,000
Fisher information sample size	1,000
Fisher information batch size	32