A PROBABILISTIC FRAMEWORK FOR MODULAR CONTINUAL LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Continual learning (CL) algorithms seek to accumulate and transfer knowledge across a sequence of tasks and achieve better performance on each successive task. Modular approaches, which use a different composition of modules for each task and avoid forgetting by design, have been shown to be a promising direction to CL. However, searching through the large space of possible module compositions remains a challenge. In this work, we develop a scalable probabilistic search framework as a solution to this challenge. Our framework has two distinct components. The first is designed to transfer knowledge across similar input domains. To this end, it models each module's training input distribution and uses a Bayesian model to find the most promising module compositions for a new task. The second component targets transfer across tasks with disparate input distributions or different input spaces and uses Bayesian optimisation to explore the space of module compositions. We show that these two methods can be easily combined and evaluate the resulting approach on two benchmark suites designed to capture different desiderata of CL techniques. The experiments show that our framework offers superior performance compared to state-of-the-art CL baselines.

1 INTRODUCTION

The *continual learning* (CL) (Thrun & Mitchell, 1995) setting calls for algorithms that can solve a sequence of learning problems while performing better on every successive problem. A CL algorithm should avoid catastrophic forgetting — i.e., not allow later tasks to overwrite what has been learned from earlier tasks — and achieve transfer across a large sequence of problems. Ideally, the algorithm should be able to transfer knowledge across similar input distributions (*perceptual transfer*), dissimilar input distributions and different input spaces (*non-perceptual transfer*), and to problems with a few training examples (*few-shot transfer*).

Modular approaches (Valkov et al., 2018; Veniat et al., 2020; Ostapenko et al., 2021) have been proposed as a promising direction for CL. These approaches represent a neural network as a composition of modules, in which each module is a parameterised function that is trained to perform an atomic transformation of its input and is reusable across tasks. Modular CL algorithms accumulate a library of diverse modules by solving the encountered problems in a sequence. Given a new problem, their goal is to find the most suitable composition of modules, out of the set of all possible compositions, as measured by the performance on a held-out dataset. Unlike CL approaches which share the same parameters across all problems, modular algorithms can introduce new modules and, thus, do not have an upper bound on the number of solved problems.

However, *scalability* remains a key challenge in modular approaches to CL, as the set of module compositions is discrete and explodes combinatorially. Prior work has often sidestepped this challenge by introducing various restrictions on the compositions, for example, by limiting the number of modules that can be stored in memory (Ostapenko et al., 2021) or only handling perceptual transfer (Veniat et al., 2020). The design of CL algorithms that relax these restrictions and can also scale remains an open problem.

In this work, we present a new probabilistic framework for modular CL as a response to this challenge. Given a new problem, our framework computes a distribution over the available previously trained modules and then selects the module combinations with the highest probability. Our technical contributions are threefold. First, we develop a probabilistic model over module combinations

that can achieve perceptual and few-shot transfer. Second, we identify a subset of module combinations capable of non-perceptual transfer and introduce a new kernel between them, allowing us to define a probabilistic model over this subset. Finally, we show that each of the two models can be used to conduct separate searches through module combinations, which can then be combined into a scalable modular CL algorithm capable of perceptual, few-shot and non-perceptual transfer.

We conduct experiments on two benchmark suites that evaluate different aspects of CL. Our experiments show that our approach achieves different types of knowledge transfer in large search spaces, is applicable to different input domains and modular neural architectures, and outperforms competitive baselines.

2 BACKGROUND

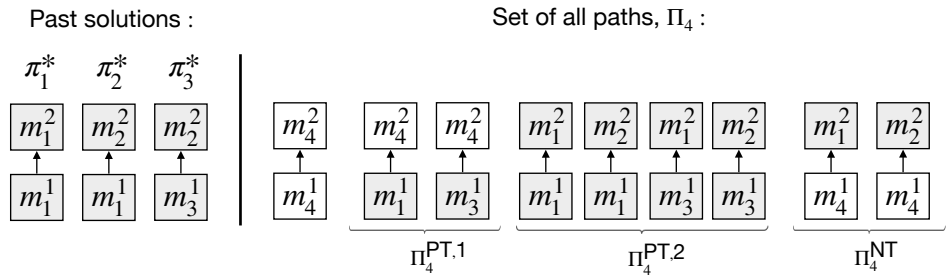


Figure 1: This illustrates the set of all paths, Π_4, which a modular algorithm considers when solving the fourth problem Ψ_4 of some sequence, where the modular architecture has $L = 2$ layers. The library consists of all previously trained modules: $\Lambda_4 = \{m_1^1, m_3^1, m_1^2, m_2^2\}$. Paths in $\Pi_4^{\text{PT},1}$ (Eq. 4) select a pre-trained module for the first layer, which allows them to perform perceptual transfer. Paths in $\Pi_4^{\text{PT},2}$ reuse modules in both layers. They can perform few-shot transfer since they only require a few examples, so that the correct path can be selected. Paths in $\Pi_4^{\text{NT},1}$ (Eq. 8) can achieve non-perceptual transfer by reusing a module in the second layer, allowing for it to be applied to a new input domain.

A continual learning algorithm $\mathcal{A}$ is tasked with solving a sequence of problems $S = (\Psi_1, ..., \Psi_T)$, usually provided one at a time. We consider the supervised setting, in which each problem is characterised by a tuple $\Psi = (\mathcal{D}, \mathcal{T})$, where $\mathcal{D}$ is the input domain, comprised of an input space and an input distribution, and $\mathcal{T}$ is a task, defined by a label space and a labelling function (Pan & Yang, 2009). A CL algorithm aims to transfer knowledge between the problems in a sequence in order to improve each problems' generalisation performance. The knowledge transfer to a single problem can be defined as the difference in performance when the rest of the problems are not available.

To maximise its performance, a CL algorithm should possess a number of desiderata. First, *plasticity* refers to an algorithm's ability to continuously learn to solve new problems. Second, *stability* is required in order to prevent *catastrophic forgetting*, which can occur when an algorithm's performance on a past problem decreases drastically. Third, *forward transfer* refers to an algorithm's ability to transfer knowledge to a newly encountered problem. We distinguish between three types of knowledge transfer: between problems with similar input distributions (*perceptual*), between problems with different input distributions or different input-spaces (*non-perceptual*) and to problems with a few training examples (*few-shot*). Fourth, a CL algorithm should be capable of *backward transfer*, allowing it to improve its performance of previously encountered problems after solving new ones. Finally, a CL algorithm should be applicable to large sequences of problems, with its resource demands scaling sub-linearly with the number of solved problems (*scalability*).

Modular approaches represent a deep neural network ζ_Θ as a composition of modules $\zeta_\Theta = m^L \circ ... \circ m^2 \circ m^1$, where the first module that is applied to the input is m^1. Each module m^i represents a nonlinear transformation, parameterised by $\theta(m^i)$. It can consist of one or more hidden layers, each with a potentially different type (e.g. convolutional or fully connected) and with a different activation function (e.g. ReLU or $tanh$). Given a problem, one needs to select optimal values for all parameters Θ and return the resulting neural network ζ_Θ as the solution.

Modular approaches have been successfully applied to continual learning. After solving $t-1$ problems, they accumulate a library of previously trained modules that each have been used to help perform one or more of the previously encountered tasks. The library $\Lambda_t = \bigcup_{i=1}^{L} \{m_j^i\}_{j<t}$ is a set of pre-trained modules, where m_j^i denotes a module in layer i that was trained on the j-th problem. It is then possible to construct different modular neural networks by either selecting a pre-trained module from the library or by training a new one from scratch for each of the L layers. We denote the set of all modular neural networks as $\Pi_t = \{(\{m_j^1\}_{j<t} \cup \{m_t^1\}) \times ... \times (\{m_j^L\}_{j<t} \cup \{m_t^L\})\}$, where m_t^i denotes new module used in layer i with randomly initialised parameters. We refer to each element of Π_t as a path, as it can be seen as guiding the forward computation through different modules. Figure 1 illustrates the set of all paths Π_4 for the fourth problem of some sequence, using a modular architecture with 2 module layers. Let a new problem Ψ_t be specified by a training and a validation datasets, $\mathbf{D}^{\text{tr}} = (\mathbf{X}^{\text{tr}}, \mathbf{Y}^{\text{tr}})$ and $\mathbf{D}^{\text{val}} = (\mathbf{X}^{\text{val}}, \mathbf{Y}^{\text{val}})$. To solve it, a modular CL algorithm searches for the path which leads to the best validation performance by optimising:

$$\pi_t^* = \underset{\pi \in \mathbb{S}(\Pi_t)}{\arg\max}\, p(\mathbf{Y}^{\text{val}} | \pi^{[\text{pre}]}, \pi^{[\text{new}]}_{\Theta^*_{\text{new}}}, \mathbf{X}^{\text{val}}) \tag{1}$$

$$\text{s.t. } \Theta^*_{\text{new}} = \underset{\Theta_{\text{new}}}{\arg\max}\, p(\mathbf{Y}^{\text{tr}} | \pi^{[\text{pre}]}, \pi^{[\text{new}]}_{\Theta_{\text{new}}}, \mathbf{X}^{\text{tr}}). \tag{2}$$

Here $\pi^{[\text{pre}]}$ and $\pi^{[\text{new}]}$ denote the pre-trained and randomly initialised modules, respectively. If a path π contains new randomly initialised modules, their parameters $\Theta^*_{\text{new}} = \{\theta(m_j^i) : m_j^i \in \pi^{[\text{new}]}\}$ need to be trained before the path can be evaluated. In this case, said parameters are trained on the training dataset and the resulting model is evaluated on the validation dataset. As a result, evaluating a path can be computationally expensive and one wants to evaluate as few paths as possible. However, the search space grows quickly. If there are J pre-trained modules for each module types in the library, the search space consists of $(J+1)^L$ unique paths. This necessitates a search strategy $\mathbb{S}(\Pi_t)$ which can prioritise the most promising paths for evaluation. Under finite resources, the algorithm can evaluate only the first r paths suggested by the search strategy.

Modular algorithms can achieve many of the CL desiderata (Valkov et al., 2018). Catastrophic forgetting is prevented by freezing the parameters of all pre-trained modules in the library. A path with only randomly initialised parameters can be selected, ensuring the methods' plasticity. As more problems are solved, the pre-trained modules in the library can be reused in order to achieve all types of forward transfer, as discussed in the following sections. However, a key challenge has been to attain all transfer using a search strategy which scales to the enormous search space.

3 Probabilistic Modular Continual Learning

We propose a probabilistic approach which uses the information available about the new problem to compute a probability distribution over different choices for pre-trained modules. The distribution does not model the new modules which allows it to be queried efficiently. This leads to a principled search strategy.

Our approach is to split Π_t into subsets $\Pi_t^i \subset \Pi_t$ of paths in which the reused modules are at the same layer positions. For each subset, one can then define a Bayesian model over the choice of pre-trained modules $p(\pi^{[\text{pre}]} | \mathbf{X}^{\text{tr}}, \mathbf{Y}^{\text{tr}}, \mathbf{X}^{\text{tr}}, \mathbf{Y}^{\text{val}}, E_j)$, in which one can make use of the available datasets and the previously evaluated paths E_j. Calculating this value can be done efficiently, as it does not does not involve randomly initialised modules. We can then define a search strategy for each path subset as selecting the unevaluated path with the highest posterior probability of its pre-trained modules:

$$\mathbb{S}_{\text{MAP}}(\Pi_t^i) = \left(\underset{\pi \in \Pi_t^i}{\arg\max}\, p(\pi^{[\text{pre}]} | \mathbf{X}^{\text{tr}}, \mathbf{Y}^{\text{tr}}, \mathbf{X}^{\text{tr}}, \mathbf{Y}^{\text{val}}, E_j) \right)_j \tag{3}$$

Multiple subsets of Π_t can be explored by applying this search strategy to each sequentially.

Next, we first apply our approach to subsets of paths which are capable of perceptual and few-shot transfer. Second, we apply it to a subset of paths capable of non-perceptual transfer. In both cases, we define a suitable Bayesian model, which takes advantage of each subset's properties. Finally, we show how the two search strategies can be combined. Developing strategies for the remaining subsets of Π_t is left for future work.

4 EFFICIENT PERCEPTUAL TRANSFER AND FEW-SHOT TRANSFER

In order to achieve perceptual transfer, a model needs to transfer knowledge on how to transform the input. For a path, this means reusing the first $l \in \{1, ..., L\}$ module layers. If all L layers are reused, this can allow for few-shot transfer, since there are no new parameters that need to be learned. Therefore, we identify subsets of paths $\Pi_t^{\text{PT},l} \in \Pi_t$ which can achieve perceptual transfer and few-shot transfer. In each path, the first l layers are selected from the library, while the rest are selected to be new randomly initialised modules, i.e.

$$\Pi_t^{\text{PT},l} = \{\pi : \pi = \{m_{<t}^i, i \leq l\} \cup \{m_t^i, i > l\}\}. \tag{4}$$

Each subset grows polynomially with the size of the library and exponentially with the number of layers, which makes a naive search inapplicable when either of the two quantities is large. In this section we use our probabilistic framework to devise an efficient search strategy.

4.1 PROBABILISTIC MODEL

In general, if a neural network is evaluated on inputs which not sampled from its training distribution, its performance can degrade significantly (Csurka, 2017). We hypothesise that the density of an input under the model's training input distribution is indicative of how well the model can transform the input. We extend this idea to modules since they are parameterised nonlinear transformations. Therefore, for a new problem and a given path, the density of each pre-trained module's inputs under said module's input distribution should be indicative of the path's expected performance. For a PT path, the first l modules are pre-trained, which allows us to use them to process the training data and calculate the inputs which each of these modules needs to handle.

Using this insight, for a given value of l, we define a probabilistic model over the choices of pre-trained modules and their N inputs (including X - the problem inputs, and H^i - relevant hidden states):

$$\begin{aligned} &p(M^1, ..., M^l, X_1, ..., X_N, H_1^1, ..., H_N^1, ..., H_1^{l-1}, ..., H_N^{l-1}) \\ &\qquad = \prod_{j=1}^{N} \left\{ p(X_j|H_j^1, M^1) \prod_{i=2}^{l-1} \left[p(H_j^{i-1}|H_j^i, M^i)\right] p(H_j^{l-1}|M^l) \right\} \prod_{i=1}^{l} \left[p(M^i)\right]. \end{aligned} \tag{5}$$

We then model the posterior of a PT path of length l as only being dependent on the training inputs, $p(\pi^{[\text{pre}]}|\mathbf{X}^{\text{tr}}) := p(m^1, ..., m^l|\mathbf{x}_1, ..., \mathbf{x}_N)$, and express it in terms of quantities which we can approximate, as derived in Appendix A:

$$\begin{aligned} &p(m^1, ..., m^l|\mathbf{x}_1, ..., \mathbf{x}_N) \propto p(m^1, ..., m^l, \mathbf{x}_1, ..., \mathbf{x}_N) \\ &\qquad \approx \prod_{j=1}^{N} \left\{ \prod_{i=1}^{l-1} \left[\frac{q(\mathbf{h}_j^{i-1}|m^i)}{\sum_{m^{i+1'}} q(\mathbf{h}_j^i|m^{i+1'}) p(m^{i+1'})} \right] q(\mathbf{h}_j^{l-1}|m^l) \right\} \prod_{i=1}^{l} p(m^i) \end{aligned} \tag{6}$$

We define a prior distribution over pre-trained modules which gives preference to modules which helped achieve a higher accuracy, on the problems which they have been trained on. Our motivation is that such modules have learned a more generalisable transformation of their inputs. We use softmax to define a prior, $p(M^i)$, over the modules in layer i which is proportional to each module's previously achieved accuracy.

We set out to construct an approximation of a module's training input distribution, $p(H^{i-1}|M^i)$, which is easy to compute in terms of implementation, storage and computational resources. After a module is trained, we perform dimensionality reduction of each sample from the module's training distribution. We use a random projection (Johnson, 1984) to reduce each sample's dimensionality from d to $k \ll d$. Finally, we compute the sample mean and covariance of the projected data sample in order to approximate their distribution using a multivariate Gaussian. This greatly reduces the number of parameters required to approximate the resulting distribution from $d + d^2$ to $k + k^2$. When maximising the posterior using Eq. 6, we compute $q(\mathbf{h}_j^{i-1}|m^i) \propto q(\mathbf{A}\mathbf{h}_j^{i-1}|m^i)$.

4.2 SEARCH STRATEGY

The number of possible PT paths of length l increase exponentially with l. As a result, it is not computationally feasible to evaluate $p(\pi|\mathbf{X}^{\text{tr}})$ for all PT paths in order to directly apply the $\mathbb{S}_{\text{MAP}}$ search strategy (Eq. 11). Instead, we augment it with a greedy policy, such that after selecting the first l pre-trained modules, we freeze this selection and reuse the same l modules for PT paths which transfer more modules. Moreover, for each value of l, the search only recommends the path which has the highest probability under our model, since this is the path which is best equipped to handle the inputs. We define the augmented search strategy over all Π^{PT} as:

$$\begin{aligned}
&\mathbb{S}_{\text{G}}^{\text{PT}}(\Pi_t^{\text{PT}}) := \{\pi^{*\text{PT},l}\}_{l=1}^{L} \\
&\text{where } \pi^{*\text{PT},l} = \pi^{*\text{PT},l-1}[:l-1] \cup m^{*,l} \cup \{m_t^i\}_{i=l}^{L} \\
&\qquad m^{*,l} = \arg\max_{m^l} p(\pi^{*'\text{PT},l-1}[:l-1] \cup m^l|\mathbf{x}_1, ..., \mathbf{x}_N)
\end{aligned} \tag{7}$$

The storage requirement of this search is associated with the number of modules in the library. Therefore, it scales linearly with the number of solved problems, $\mathcal{O}(t)$. The exact number depends on the type of similarity between the encountered problems. If perceptual transfer and few-shot transfer is often possible, the number of modules in the library will grow more slowly. Our greedy search strategy requires that each module from the library is loaded at most once, which leads to a constant memory requirement. The computational complexity of this search remains constant with the number of solved problems since it selects a constant number of paths for each problem: $L + 1$ (L suggested by our search strategy, plus a randomly initialised model).

5 EFFICIENT NON-PERCEPTUAL TRANSFER

To achieve non-perceptual transfer, an algorithm can use pre-trained modules for the last l module layers. This represents knowledge on how to transfer a latent representation of the input to a task-specific prediction. Therefore, we identify subsets $\Pi_t^{NT,l} \in \Pi_t$ which consists of paths capable of non-perceptual transfer. Each path has its first $L - l$ randomly initialised while the rest l modules are selected from a library:

$$\Pi_t^{\text{NT},l} = \left\{\pi_t^{\text{NT},l} : \pi_t^{\text{NT},l} = \{m_t^i\}_{i=1}^{l} \cup \{m_{<t}^i\}_{i=L-l+1}^{L}\right\}. \tag{8}$$

5.1 PROBABILISTIC MODEL

For NT paths of length l, we model the posterior over the choice of pre-trained modules as being dependent on the validation set and the past evaluations and express it as:

$$p(\pi^{[\text{pre}]}|\mathbf{X}^{\text{val}}, \mathbf{Y}^{\text{val}}, \mathbf{E}_j) \propto p(\mathbf{Y}^{\text{val}}|\mathbf{X}^{\text{val}}, \pi^{[\text{pre}]}, \mathbf{E}_j)p(\pi^{[\text{pre}]}). \tag{9}$$

We define a prior over $\pi^{[\text{pre}]}$ which assigns equal non-zero values only to pre-trained modules which have been used together to solve a previous problem. This reflects our prior assumption that using novel combination of modules for non-perceptual transfer is unnecessary for the sequences which we consider.

We model $p(\mathbf{Y}^{\text{val}}|\mathbf{X}^{\text{val}}, \pi^{[\text{pre}]}, \mathbf{E}_j)$ using a Gaussian process (GP), which is fit on previous evaluations E_j. To enable this, we next define a kernel function between the pre-trained modules of two NT paths.

The difference between two NT paths π_i and π_j of length l is that their last l pre-trained modules compute different functions. We hypothesise that if said functions are similar, the two NT paths will exhibit similar performance after their random parameters are trained. To make use of this insight, we define an inner product between two vector-valued functions, $f : \Omega \to \mathbb{R}^r$ and $g : \Omega \to \mathbb{R}^r$ as: $\langle f\ ,\ g\rangle = \int_\Omega f(\mathbf{z}) \cdot g(\mathbf{z}) d\mathbf{z}$. This allows us to compute the distance between two functions as:

$$d(f, g) := ||f - g|| = \sqrt{\langle f - g\ ,\ f - g\rangle} = \sqrt{\int_\Omega (f(\mathbf{z}) - g(\mathbf{z})) \cdot (f(\mathbf{z}) - g(\mathbf{z})) d\mathbf{z}}\ .$$

We approximate this value using Monte Carlo integration with a set of inputs from the functions' common input space. We can then define the kernel function between the chosen pre-trained modules of two same-length NT paths using the squared exponential kernel function and the distance between the functions computed by their last l modules:

$$\kappa(\pi_i^{[\text{pre}]}, \pi_j^{[\text{pre}]}; Z) = \sigma^2 \exp\left\{-(d(\pi_i^{[\text{pre}]}, \pi_j^{[\text{pre}]}; Z)^2)/(2\gamma^2)\right\} \tag{10}$$

where σ and γ are the kernel hyperparameters which are fit to maximise the marginal likelihood of a GP's training data (Rasmussen & Williams, 2006).

To create Z for a value of l, we store a small number of samples from the input distribution of each pre-trained module at layer $L-l+1$. Given a new problem, we create Z by combining all the stored hidden activations. This allows us to compare functions on regions of the input space with higher density.

5.2 Search Strategy

We modify $\mathbb{S}_{\text{MAP}}$ and apply it on a single subset $\Pi_t^{\text{NT},l_{\text{min}}}$ where l_{min} is a user-defined the minimum number of transfer which need to be transferred in order to improve the performance. Due to our choice of prior, maximising the posterior using Eq. 9 equals to maximising the likelihood over paths with nonzero probability. However, our GP-based approximation provides a distribution of the likelihood's value, which reflects the uncertainty of the GP's prediction. To account for this, we use an acquisition function, namely Upper Confidence Bound (UCB) (Shahriari et al., 2015) which provides an optimistic prediction of the likelihood. The resulting search strategy resembles Bayesian Optimisation:

$$\mathbb{S}_{\text{BO-MAP}}(\Pi_t^{\text{NT},l_{\text{min}}}) = \left(\underset{\pi \in \Pi_t^i}{\arg\max}\, UCB\left(p(\mathbf{Y}^{\text{val}}|\mathbf{X}^{\text{val}}, \pi^{[\text{pre}]}, \mathbf{E}_j)\right)\right)_j . \tag{11}$$

Our model makes it possible to detect when further improvement is unlikely, allowing us to perform early stopping. For this purpose we compute the Expected Improvement, $EI\left(p(\pi^{[\text{pre}]}|\mathbf{X}^{\text{val}}, \mathbf{Y}^{\text{val}}, \mathbf{E}_j)\right)$ of the path selected for evaluation at each step j. If it's lower than a certain threshold, we exit the search and do not recommend any more paths for evaluation. EI-based early stopping has been previously suggested in Nguyen et al. (2017) and, similarly to Makarova et al. (2022), our preliminary experiments showed that it leads to fewer evaluations, compared to using LCB for early stopping.

In the worst case $\mathbb{S}_{\text{BO-MAP}}$ would propose $\mathcal{O}(t-1)$ paths for evaluation which scales linearly with the number of solved problems. While early stopping reduces the number of evaluated paths, our experiments suggest said number still scales linearly with the number of problems. However, our ablation experiments, presented in Appendix D, demonstrate that our search strategy exhibits a good anytime performance. This means that our Bayesian optimisation algorithm is capable of quickly finding a path which achieves positive non-perceptual transfer. Therefore, it is possible to constrain $S_{\text{BO-MAP}}$ to evaluating a constant number of paths and still achieve non-perceptual transfer across a long sequence of problems.

6 Experiments

Our experiments evaluate our approaches' perceptual, non-perceptual and few-shot transfer capabilities, and assess the algorithms' ability to scale to a long sequence of problems. Furthermore, we assess their applicability to disparate input domains and neural architectures. For this purpose, we perform experiments on a new benchmark suite as well as on the CTrL (Veniat et al., 2020) benchmark suite.

We derive three modular CL algorithms: PeCLwhich can perform perceptual and few-shot transfer using $\mathbb{S}_{\text{G}}^{\text{PT}}(\Pi_t^{\text{PT}})$ (Eq 7); NoCLwhich can perform non-perceptual transfer using $\mathbb{S}_{\text{BO-MAP}}(\Pi_t^{\text{NT},l_{\text{min}}})$ (Eq. 11); CCLwhich combines both search strategies using $\mathbb{S}_{\text{CCL}}(\Pi_t^{\text{PT}} \cup \Pi_t^{\text{NT},l_{\text{min}}}) := S_{\text{G}}^{\text{PT}}(\Pi_t^{\text{PT}}) + S_{\text{BO-MAP}}(\Pi_t^{\text{NT},l_{\text{min}}})$. We compare our algorithms to a number of competitive modular CL baselines:

Standalone (SA) baseline which trains a new model for every problem, MCL-RS which randomly selects paths from Π_t (shown to be a competitive baseline in high-dimensiaonl search spaces and in neural architecture search (Li & Talwalkar, 2020)); HOUDINI(Valkov et al., 2018) with a fixed neural architecture to keep the results comparable; and MNTDP-D (Veniat et al., 2020).

We use the same set of hyperparameters, listed in Appendix C, for our approaches across both benchmark suite, which suggests that this choice of hyperparameters is robust and applicable to different problems and neural architectures. Moreover, we use a problem-specific random seed to make the training process deterministic, so that the difference in performance can be accredited only to the LML algorithm, and not to randomness introduced during training.

For each baseline, we assess the performance on a held-out test dataset and report the average accuracy of the final model across all problems, $\mathcal{A}$, as well as the amount of forward transfer on the last problem, Tr^{-1}, computed as the difference in accuracy, compared to the standalone baseline.

6.1 Compositional Benchmarks

We introduce a benchmark suite consisting of different sequences which can diagnose different CL properties. We build on the CTrL benchmarks (Veniat et al., 2020) by instead using problems with compositional tasks, in which each problem combines an image classification task with a two-dimensional pattern recognition task. This allows us to introduce new sequences which evaluate more properties, namely non-perceptual transfer across input spaces and few-shot transfer, as well as to introduce more challenging versions of the sequences which evaluate perceptual transfer. We have the following sequences of length 6. In S^{pl}, each problem is represented by a large dataset in order to evaluate an algorithm's plasticity. In S^{+} the last problem is the same as the first, but is represented with a larger dataset, allowing for backward transfer to be evaluated. Conversely, in S^{-}, the first and last problems are also the same but the first is represented by a bigger dataset, thus, evaluating forward knowledge transfer to the same problem. We define three different sequences which evaluate perceptual transfer. In S^{out}, the input domains of the first and last problems are the same. In $S^{\text{out}*}$ the input domains of the first, second and last problems are the same, making it harder to decide which problem to transfer from. In $S^{\text{out}**}$ the first and last problems are the same, however, perceptual knowledge needs to be transferred from the second problem, as it also has the same domain but is represented by a larger dataset. S^{few} evaluates few-shot transfer, since the last problem is represented by 10 data points and solving it requires re-combining knowledge from problems 2 and 4 in a novel way. Finally, there are two sequences evaluating non-perceptual transfer. In S^{in} the first and last problem share the same two-dimensional pattern, while having different input image domains. S^{sp} is the same as S^{in}, except last problem's input space is also re-shaped from an image to a one-dimensional vector, which loses the structural information and requires different architecture for the first few modules in order to process the input. Therefore, this evaluate an algorithm's ability to transfer across different input spaces and similar but not identical neural architectures. Finally, S^{long} is a sequence of 60 randomly selected problems, which evaluates perceptual and non-perceptual transfer on long sequences.

The neural architecture which we use consists of $L = 8$ modules, which leads to a large search space, since even for $t = 6$, there are $\mathcal{O}(t^L = 1679616)$ different possible paths. Full specification of the problems, sequences, neural architecture and training procedure can be found in Appendix C. We create 3 realisations of each sequence by randomly selecting different compositional problems. Afterwards, for each sequence, we report the measurements, averaged over these 3 versions.

6.1.1 Results

PeCL and CCL exhibit the same performance for sequences which do not involve non-perceptual transfer, as presented in Table 1. Our method demonstrates similar plasticity to the others, according to S^{pl}, as well as similar performance on S^{+} which evaluates backward transfer. It can be observed that $\mathbb{S}_G^{\text{PT}}$ enables our algorithms to outperform the baselines on the sequences which require forward knowledge transfer, with the other scalable modular algorithm, MNTDP-D, being second. On $S^{\text{out}**}$, our performance-based prior helps our method identify the correct modules to be transferred, leading to a $+10.3$ higher accuracy on the last problem than MNTDP-D. On S^{few} our algorithm's ability to perform few-shot transfer leads to $+34.7$ higher accuracy on the last problem than MNTDP-D.

	SA	MCL-RS	HOUDINI	MNTDP-D	PeCL, CCL
$\mathcal{A}(S^{-})$	73.88	76.67	79.59	81.67	**81.92**
$Tr^{-1}(S^{-})$	0.	17.22	34.27	**34.29**	**34.29**
$\mathcal{A}(S^{\text{out}})$	74.25	76.16	74.40	77.95	**78.15**
$Tr^{-1}(S^{\text{out}})$	0.	5.64	0.	**15.41**	**15.41**
$\mathcal{A}(S^{\text{out*}})$	72.27	73.39	72.27	75.48	**75.72**
$Tr^{-1}(S^{\text{out*}})$	0.	0.43	0.	**12.53**	**12.53**
$\mathcal{A}(S^{\text{out**}})$	71.51	73.85	71.75	73.71	**75.73**
$Tr^{-1}(S^{\text{out**}})$	0.	4.61	1.46	1.74	**12.04**
$\mathcal{A}(S^{\text{few}})$	75.47	78.14	80.82	82.18	**88.12**
$Tr^{-1}(S^{\text{few}})$	0.	5.87	4.54	11.42	**46.07**
$\mathcal{A}(S^{\text{pl}})$	93.61	93.63	93.61	93.72	**93.79**
$Tr^{-1}(S^{\text{pl}})$	0.	0.	0.	**00.20**	**00.20**
$\mathcal{A}(S^{+})$	73.61	**75.08**	73.61	74.54	74.49
$Tr^{-1}(S^{+})$	0.	0.	0.	0.	0.

Table 1: The results on our compositional benchmarks which do not evaluate non-perceptual transfer.

	SA	MCL-RS	HOUDINI	MNTDP-D	PeCL	NoCL	CCL
$\mathcal{A}(S^{\text{in}})$	89.01	90.85	89.32	90.62	90.26	92.20	**92.82**
$Tr^{-1}(S^{\text{in}})$	0.	1.81	11.04	9.70	7.61	18.89	**22.28**
$\mathcal{A}(S^{\text{sp}})$	87.94	92.22	**92.99**	87.94	87.92	91.92	91.93
$Tr^{-1}(S^{\text{sp}})$	0.	25.68	**30.27**	0.	0.	23.65	23.65

Table 2: The results on our compositional benchmarks which evaluate non-perceptual transfer.

Our results on sequences which evaluate non-perceptual transfer, presented in Table 2, demonstrate that $\mathbb{S}_{\text{BO-MAP}}$ enables NoCLand CCLto transfer knowledge across different input distributions and input spaces. In S^{sp}, the different input space necessitates a different modular architecture for the first 5 modules, resulting in a much smaller search space, $\mathcal{O}(6^3 = 216)$. This allows non-scalable approaches, namely MCL-RS and HOUDINI, to also be effective on this sequence. Lastly, CCL's performance demonstrates that combining $\mathbb{S}_G^{\text{PT}}$ and $\mathbb{S}_{\text{BO-MAP}}$ leads to a better performance on sequences that allow for both perceptual and non-perceptual transfer.

Finally we evaluated our approach on S^{long} - a sequence of 60 problems. PeCL achieved $+7.37$ higher average accuracy than the standalone baseline demonstrating its ability to achieve perceptual transfer on a long sequence of problems. MNTDP-D achieved 8.83 higher average accuracy than SA which confirmed its scalability. Finally, CCL performed the best, attaining $+12.25$ higher average accuracy than SA. This shows that our approach can successfully attain perceptual and non-perceptual transfer across a long sequence of problems.

6.2 CTrL BENCHMARKS

The CTrL benchmark suite (Veniat et al., 2020) defines fewer sequences to evaluate different CL properties. Namely, they specify S^{pl}, S^{+}, S^{-}, S^{out}, S^{sp} which are defined identically to ours. In contrast, the sequences are over multi-class classification tasks of coloured images from different domains. They also use a different modular architecture, based on ResNet18 (He et al., 2016), which is more complex than the architecture used in the previous benchmarks. Our experimental setup, detailed in Appendix C, mirrors the one used in Veniat et al. (2020), except that we make the training process deterministic, as discussed above.

Our results, shown in Table 3, demonstrate that our approach, namely PeCL and CCL, achieves similar performance to MNTDP-D on S^{pl}, S^{+}, S^{-}, S^{out}, S^{sp} which evaluate plasticity, backward and perceptual transfer. On S^{-}, NoCL and CCL successfully perform non-perceptual transfer, leading to superior performance on the last problem of the sequence ($+10.45$ higher than MNTDP). CTrL also specifies S^{long} which has 100 problems but only evaluates perceptual transfer. PeCL was successfull in transferring knowledge, achieving an average accuracy that was $+14.48$ higher than SA.

	SA	MNTDP-D	PeCL	NoCL	CCL
$\mathcal{A}(S^{\text{in}})$	58.77	61.36	61.78	**63.41**	63.10
$Tr^{-1}(S^{\text{in}})$	0.	22.12	24.67	**32.57**	**32.57**
$\mathcal{A}(S^{-})$	56.28	81.67	**81.92**	-	**81.92**
$Tr^{-1}(S^{-})$	0.	**34.29**	**34.29**	-	**34.29**
$\mathcal{A}(S^{\text{out}})$	74.25	77.95	**78.15**	-	**78.15**
$Tr^{-1}(S^{\text{out}})$	0.	**15.41**	**15.41**	-	**15.41**
$\mathcal{A}(S^{\text{pl}})$	58.25	93.72	**93.79**	-	**93.79**
$Tr^{-1}(S^{\text{pl}})$	0.	**00.20**	**00.20**	-	**00.20**
$\mathcal{A}(S^{+})$	73.61	**74.54**	74.49	-	74.49
$Tr^{-1}(S^{+})$	0.	0.	0.	-	0.

Table 3: The evaluations on the CTrL sequences, except for S^{long}. For each sequence, we report average accuracy $\mathcal{A}$ and the amount of forward transfer on the last problem Tr^{-1}.

Overall, the results demonstrates that both $\mathbb{S}_{\text{BO-MAP}}$ are also applicable to more complex modular architectures.

7 Related Work

This work considers the data-incremental (De Lange et al., 2021) supervised setting of continual learning. Other settings can involve overlapping problems Farquhar & Gal (2018) or reinforcement learning (Khetarpal et al., 2020). CL desidarata is also presented in Schwarz et al. (2018); Hadsell et al. (2020); Delange et al. (2021), which include other properties, e.g. selective forgetting of trivial information, but do not distinguish between different types of forward transfer. We list such types following Valkov et al. (2018), but add the requirement of non-perceptual transfer over input spaces.

Continual learning methods can be categorised into ones based on regularisation, replay or a dynamic architecture (Parisi et al., 2019). The first two share the same set of parameters across all problems which limits their capacity and, in turn, their plasticity (Kirkpatrick et al., 2017). Dynamic architecture methods can share different parameters by learning a problem-specific parameter masks (Mallya & Lazebnik, 2018) or adding more parameters (Rusu et al., 2016). This category includes modular approaches to CL, which share and introduce new modules, allowing groups of parameters to be trained and always reused together. Modular approaches mainly differ by their search space and their search strategy. PathNet (Fernando et al., 2017) uses evolutionary search to search through paths that combine up to 4 modules per layer. Rajasegaran et al. (2019) use random search on the set of all paths. HOUDINI (Valkov et al., 2018) uses type-guided exhaustive search on the set of all possible modular architectures and all paths. This method can attain the three types of forward transfer, but does not scale to large search spaces. Their results also show that exhaustive search leads to better performance than evolutionary search. MNTDP-D (Veniat et al., 2020) is a scalable approach which restrict its search space to perceptual transfer paths, derived from previous solutions. Similarly to $\mathbb{S}_G^{\text{PT}}$, MNTDP-D evaluates only $L+1$ paths per problem, however, the approach does not allow for novel combinations of pre-trained modules which prevents it from achieving few-shot transfer. In contrast, our approach can achieve all three types of forward transfer. LMC (Ostapenko et al., 2021) makes a soft selection over paths. For each layer, they compute a linear combination of the outputs of all available pre-trained modules. To do this, the authors model the distribution over each module's outputs using a separate auto-encoder. In contrast, for $\mathbb{S}_{\text{G}}^{\text{PT}}$ we use multivariate Gaussian approximations of the projected inputs of a pre-trained module, which requires orders of magnitude fewer parameters. LMC can achieve perceptual and few-shot transfer, but is not capable of non-perceptual transfer. Finally, LMC requires that all modules be kept in memory which limits it applicability to larger libraries, thus, larger search spaces.

8 Conclusion

We introduced a probabilistic framework for modular continual learning and showed that it can be used to develop a scalable approach, capable of perceptual, few-shot and non-perceptual transfer.

REFERENCES

Abalone. UCI Machine Learning Repository, 1995.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and M Ranzato. Continual learning with tiny episodic memories. 2019.

Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3606–3613, 2014.

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017. doi: 10.1109/ijcnn.2017.7966217.

Gabriela Csurka. Domain adaptation for visual applications: A comprehensive survey. *arXiv preprint arXiv:1702.05374*, 2017.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021.

Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Sebastian Farquhar and Yarin Gal. Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*, 2018.

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.

Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. Online meta-learning. In *International Conference on Machine Learning*, pp. 1920–1930. PMLR, 2019.

GPy. GPy: A gaussian process framework in python. `http://github.com/SheffieldML/GPy`, since 2012.

Raia Hadsell, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in cognitive sciences*, 24(12):1028–1040, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

William B Johnson. Extensions of lipschitz mappings into a hilbert space. *Contemp. Math.*, 26: 189–206, 1984.

Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *arXiv preprint arXiv:2012.13490*, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, pp. 367–377. PMLR, 2020.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Anastasia Makarova, Huibin Shen, Valerio Perrone, Aaron Klein, Jean Baptiste Faddoul, Andreas Krause, Matthias Seeger, and Cedric Archambeau. Automatic termination for hyperparameter optimization. In *First Conference on Automated Machine Learning (Main Track)*, 2022.

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

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Vu Nguyen, Sunil Gupta, Santu Rana, Cheng Li, and Svetha Venkatesh. Regret for expected improvement over the best-observed value and stopping condition. In *Asian conference on machine learning*, pp. 279–294. PMLR, 2017.

Oleksiy Ostapenko, Pau Rodriguez, Massimo Caccia, and Laurent Charlin. Continual learning via local module composition. *Advances in Neural Information Processing Systems*, 34:30298–30312, 2021.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Jathushan Rajasegaran, Munawar Hayat, Salman Khan, Fahad Shahbaz Khan, and Ling Shao. Random path selection for incremental learning. *Advances in Neural Information Processing Systems*, 3, 2019.

Carl Edward Rasmussen and Christopher KI Williams. Gaussian processes for machine learning. 2006.

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.

Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pp. 4528–4537. PMLR, 2018.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2015.

Sebastian Thrun and Tom M Mitchell. Lifelong robot learning. *Robotics and autonomous systems*, 15(1-2):25–46, 1995.

Lazar Valkov, Dipak Chaudhari, Akash Srivastava, Charles Sutton, and Swarat Chaudhuri. Houdini: Lifelong learning as program synthesis. *Advances in Neural Information Processing Systems*, 31, 2018.

Tom Veniat. Mntdp. `https://github.com/TomVeniat/MNTDP`, 2021.

Tom Veniat and Marc'Aurelio Ranzato. Continual transfer learning benchmark. `https://github.com/facebookresearch/CTrLBenchmark`, 2021.

Tom Veniat, Ludovic Denoyer, and Marc'Aurelio Ranzato. Efficient continual learning with modular networks and task-driven priors. *arXiv preprint arXiv:2012.12631*, 2020.

Ziyu Wang, Masrour Zoghi, Frank Hutter, David Matheson, Nando De Freitas, et al. Bayesian optimization in high dimensions via random embeddings. In *IJCAI*, pp. 1778–1784, 2013.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

A DERIVING THE POSTERIOR

In this section we show how we approximate the posterior distribution $p(m^1_{<t}, ..., m^l_{<t}|\mathbf{x}_1, ..., \mathbf{x}_N)$. To ease the presentation, and without loss of generality, we set the number of pre-trained modules $l = 3$. Next, we express the joint distribution in terms of quantities which we can approximate.

$$p(M^1, M^2, M^3, X_1, ..., X_N, H^1_1, ..., H^1_N, H^2_1, ..., H^2_N)$$
$$= p(M^1)p(M^2)p(M^3)\prod_{j=1}^{N} p(X_j|H^1_j, M^1)p(H^1_j|H^2_j, M^2)p(H^2_j|M^3)$$

Here, $p(X_j|H^1_j, M^1)$ can be expressed as:

$$p(X_j|H^1_j, M^1) = \frac{p(X_j, H^1_j, M^1)}{p(H^1_j, M^1)} = \frac{p(H^1_j|X_j, M^1)p(X_j|M^1)p(M^1)}{p(H^1_j)p(M^1)} = \frac{p(H^1_j|X_j, M^1)p(X_j|M^1)}{\sum_{m^{2'}} p(H^1_j|m^{2'})p(m^{2'})}.$$

Moreover, $p(H^1_j|H^2_j, M^2)$ can be expressed as:

$$p(H^1_j|H^2_j, M^2) = \frac{p(H^1_j, H^2_j, M^2)}{p(H^2_j, M^2)} = \frac{p(H^2_j|H^1_j, M^2)p(H^1_j|M^2)p(M^2)}{p(H^2_j)p(M^2)}$$
$$= \frac{p(H^2_j|H^1_j, M^2)p(H^1_j|M^2)}{\sum_{m^{3'}} p(H^2_j|m^{3'})p(m^{3'})}.$$

Therefore, the joint distribution can be expressed as:

$$p(M^1, M^2, M^3, X_1, ..., X_N, H^1_1, ..., H^1_N, H^2_1, ..., H^2_N)$$
$$= p(M^1)p(M^2)p(M^3)\prod_{j=1}^{N} \frac{p(H^1_j|X_j, M^1)p(X_j|M^1)}{\sum_{m^{2'}} p(H^1_j|m^{2'})p(m^{2'})} \frac{p(H^2_j|H^1_j, M^2)p(H^1_j|M^2)}{\sum_{m^{3'}} p(H^2_j|m^{3'})p(m^{3'})} p(H^2_j|M^3). \tag{12}$$

B BELL - BENCHMARKS FOR LIFELONG LEARNING

We now introduce *BELL* - a suite of benchmarks for evaluating the aforementioned LML properties. As above, we assume compositional tasks and then generate various lifelong learning sequences, with each evaluating one or two of the desired properties. Running an LML algorithm on all sequences then allows us to asses which properties are present and which are missing. This builds upon the CTrL benchmark suite, presented in Veniat et al. (2020), which defines different sequences of image classification tasks, namely S^{pl}, S^{-}, S^{out}, S^{in}, S^{+} and S^{long}. They evaluate plasticity, perceptual transfer, non-perceptual transfer, catastrophic forgetting, backward transfer and scalability. We define these sequences similarly but for problems with compositional tasks. This allows us to introduce new sequences which evaluate new LML properties (S^{sp} and S^{few}). We also introduce new more challenging sequences (S^{out*} and S^{out**}).

We assume compositional tasks and represent each problem as a triple $\Psi_t = (D_j, h_j, g_k)$ where D_j is the distribution of the inputs and h_j and g_k constitute the labelling function, as given in eq. **??**. We refer to h_j as the lower labelling sub-function and to g_k as the upper labelling sub-function. We use the indices j and k to indicate whether the corresponding labelling sub-function has occurred before in the sequence ($j < t, k < t$) or if it is new and randomly selected ($j = t, k = t$). For brevity, if $j = t$ and $k = t$, we don't write out the whole triple but only Ψ_t. By repeating previously encountered domains or labelling sub-functions, we can control what knowledge can be transferred in each of the define sequences. In turn, this allows us to evaluate different LML properties. We use Ψ^+ to indicate that the dataset generated for this problem is sufficient to learn a well generalising approximation

Sequence	Sequence Pattern	Properties
S^{pl}	$[\Psi_1^+, \Psi_2^+, \Psi_3^+, \Psi_4^+, \Psi_5^+, \Psi_6^+]$	1., 2.
S^{-}	$[\Psi_1^+, \Psi_2^-, \Psi_3^-, \Psi_4^-, \Psi_5^-, \Psi_1^-]$	3.
S^{out}	$[\Psi_1^+, \Psi_2^-, \Psi_3^-, \Psi_4^-, \Psi_5^-, \Psi_6^- = (D_1, h_1, g_6)]$	3.1
S^{out*}	$[\Psi_1^-, \Psi_2^+ = (D_1, h_1, g_2), \Psi_3^-, \Psi_4^-, \Psi_5^-, \Psi_6^- = (D_1, h_1, g_6)]$	3.1
S^{out**}	$[\Psi_1^-, \Psi_2^+ = (D_1, h_1, g_2), \Psi_3^-, \Psi_4^-, \Psi_5^-, \Psi_1^-]$	3.1
S^{in}	$[\Psi_1^+, \Psi_2^-, \Psi_3^-, \Psi_4^-, \Psi_5^-, \Psi_6^- = (D_6, h_6, g_1)]$	3.2.1
S^{sp}	$[\Psi_1^+, \Psi_2^-, \Psi_3^-, \Psi_4^-, \Psi_5^-, \Psi_6^- = (D_6, h_6, g_1)]$	3.2.2
S^{few}	$[\Psi_1^+ = (D_1, h_1), \Psi_2^+ = (D_2, h_2), \Psi_3^-, \Psi_4^- = (D_1, h_1, g_4), \Psi_5^-, \Psi_6^{--} = (D_2, h_2, g_4)]$	3.3
S^{+}	$[\Psi_1^-, \Psi_2^-, \Psi_3^-, \Psi_4^-, \Psi_5^-, \Psi_1^+]$	4.
S^{long}	$[\Psi_i]_{i=1}^{100}$	5.

Table 4: A list of all of the different LML sequences in *BELL*, each of which evaluates different LML properties. The first column contains the sequence's name, the second shows the sequence's pattern and the third column lists the LML properties evaluated by this sequence.

without transferring knowledge. On the other hand, Ψ^- indicates that the LML algorithm cannot achieve good generalisation on this problem without transferring knowledge. Finally, Ψ^{--} indicates that the generated training dataset consists of only a few datapoints, e.g. 10.

A complete list of the different sequences in *BELL* is presented in Table 4. Following Veniat et al. (2020), we set the sequence length of most sequences to 6 which, as we show in the experiments section, is sufficient for evaluating different LML properties. Next, we separately present each sequence, detailing which LML properties it evaluates.

Plasticity and **Stability**: The sequence $S^{pl} = [\Psi_1^+, \Psi_2^+, \Psi_3^+, \Psi_4^+, \Psi_5^+, \Psi_6^+]$ consists of 6 distinct problems, each of which has a different input domain and a different task. Moreover, each of the generated datasets has a sufficient number of data points as not to necessitate transfer. Therefore, this sequence evaluates a LML algorithm's ability to learn distinct problems, i.e. its plasticity (1.). Moreover, this sequence can be used to evaluate an algorithm's stability (2.) by assessing its performance after training on all problems and checking for forgetting.

Forward Transfer: Most of our sequences are dedicated to evaluate different types of forward transfer. To begin with, in the sequence $S^- = [\Psi_1^+, \Psi_2^-, \Psi_3^-, \Psi_4^-, \Psi_5^-, \Psi_1^-]$ the first and the last datasets represent the same problem, however, the last dataset has fewer data points. Therefore, an LML algorithm would need to transfer the knowledge acquired from solving the first problem, thus, demonstrating its ability to perform overall forward transfer (3.).

Perceptual Forward Transfer: We introduce three different sequences for evaluating perceptual transfer (3.1). First, in $S^{out} = [\Psi_1^+, \Psi_2^-, \Psi_3^-, \Psi_4^-, \Psi_5^-, \Psi_6^- = (D_1, h_1, g_6)]$ the last problem has the same input domain and input-processing target function h_1 as in problem 1. However, the last problem's dataset is small, therefore, an LML algorithm needs to perform perceptual transfer from the first problem, which is described by a large dataset. Second, $S^{out*} = [\Psi_1^-, \Psi_2^+ = (D_1, h_1, g_2), \Psi_3^-, \Psi_4^-, \Psi_5^-, \Psi_6^- = (D_1, h_1, g_6)]$ shares the same input distributions and lower labelling sub-function h_1 across problems Ψ_1, Ψ_2 and Ψ_6. Therefore, a LML algorithm needs to decide whether to transfer knowledge obtained from the first or from the second problem. Third, the sequence $S^{out**} = [\Psi_1^-, \Psi_2^+ = (D_1, h_1, g_2), \Psi_3^-, \Psi_4^-, \Psi_5^-, \Psi_1^-]$ is similar to the preceding one, with the distinction that the last problem is the same as the first. In this sequence, an LML algorithm needs to decide between reusing knowledge acquired from solving the same problem (Ψ_1), or to transfer perceptual knowledge from a more different problem (Ψ_2). Overall, these three sequences are designed to be increasingly more challenging in order to distinguish between different LML algorithms which are capable of perceptual transfer to a different extent.

Non-Perceptual Forward Transfer: Currently, we define two sequences to assess an algorithm's ability to transfer non-perceptual knowledge. Firstly, in $S^{in} = [\Psi_1^+, \Psi_2^-, \Psi_3^-, \Psi_4^-, \Psi_5^-, \Psi_6^- = (D_6, h_6, g_1)]$ the last problem has the same upper labelling sub-function as the first problem. However, the two problems' input distributions and lower labelling sub-functions are different. Therefore, an LML algorithm would need to transfer knowledge across different input domains (3.2.1). Secondly, the sequence $S^{sp} = [\Psi_1^+, \Psi_2^-, \Psi_3^-, \Psi_4^-, \Psi_5^-, \Psi_6^- = (D_6, h_6, g_1)]$ is simiarly defined, however, the input distribution of the last problem is also defined on a different input space from the input space of the first problem. Therefore, an algorithm would need to transfer knowledge across

different input spaces (3.2.2).

Few-shot Forward Transfer: In order to evaluate this property, we introduce the following sequence, in which the first two problems are different from the rest of the sequences: $S^{few} = [\Psi_1^+ = (D_1, h_1), \Psi_2^+ = (D_2, h_2), \Psi_3^-, \Psi_4^- = (D_1, h_1, g_4), \Psi_5^-, \Psi_6^{--} = (D_2, h_2, g_4)]$. The labelling functions of the first two problems are simpler, each consisting only of a lower labelling sub-function. This is done in order to provide a LML algorithm with more supervision on how to approximate h_1 and g_1 more accurately. The fourth problem Ψ_4 in this sequence then shares the same input domain and lower lableling sub-function as the first problem, but introduces a new upper labelling sub-function g_4. The last problem then shares the input domain and the lower labelling sub-function of Ψ_2, while also sharing the upper labelling sub-function of problem Ψ_4. Moreover, the last problem's training dataset consists of only a few data points. Therefore, a LML algorithm would need to reuse its approximations of h_2 and g_4 in a novel manner in order to solve the last problem.

Backward Transfer: The sequence $S^+ = [\Psi_1^-, \Psi_2^-, \Psi_3^-, \Psi_4^-, \Psi_5^-, \Psi_1^+]$ has the same first and last problem. However, the first dataset has significantly less data points than the last. Ideally, an LML algorithm should use the knowledge acquired after solving the last problem in order to improve its performance on the first problem. While this sequence represents a starting point for evaluating backward transfer, it is possible to introduce other sequences, representing more elaborate evaluations. For instance, introducing sequences which evaluate perceptual and non-perceptual backward transfer separately. However, as backward transfer is not the focus of this thesis, this is left for future work.

Scalability: This property can be evaluated using a long sequence of problems. For this purpose we define $S^{\text{long}} = [\Psi_i]_{i=1}^{100}$ which consists of 100 problems, each randomly selected with replacement from a set of problems. Most problems are represented by a small dataset, Ψ_t^-. Each of the first 50 problems has a $\frac{1}{3}$ probability of being represented by a large dataset, Ψ_t^+. Each problem also has a $\frac{1}{10}$ probability of being represented by an extra small dataset, Ψ_t^{--}.

The definitions of these sequences rely on three sets of domains, lower labeling sub-functions and upper labelling sub-functions, respectively. In turn, these can be used to create a set of problems. Next, we present a set of problems which can be used together with the aforementioned sequence definitions in order to evaluate LML algorithms.

B.1 Compositional Problems

To implement the sequences defined above, one needs to define a set of compositional problems. To this end, we define 9 different pairs of an input domain and a lower labelling sub-function, $\{(D_i, h_i)\}_{i=1}^{9}$. Moreover, we define 16 different upper labelling sub-functions $\{g_i\}_{i=1}^{16}$. These can be combined into a total of 144 different compositional problems.

First, we define 9 image multi-class classification tasks, which all share input and output spaces $\mathbb{R}^{28\times 28} \rightarrow \mathbb{R}^8$, but each have a different input distribution D_i and a domain-specific labelling function h_i. Concretely, we start with the following image classification datasets: MNIST (LeCun et al., 2010), Fashion MNIST (FMNIST) (Xiao et al., 2017), EMNIST (Cohen et al., 2017) and Kuzushiji49 (KMNIST) (Clanuwat et al., 2018). Since some of the classes in KMNIST have significantly fewer training data points, we only use the 33 classes with the following indices: $[0, 1, 2, 4-12, 15, 17-21, 24-28, 30, 34, 35, 37-41, 46, 47]$, as they have a sufficient number of associated data points. We split the image datasets into smaller 8-class classification datasets. We use i to denote the different splits of the same original dataset. For instance EMNIST_2 represents the third split of EMNIST, corresponding to a classification task among the letters form 'i' to 'p'. Using this we end up with the following 9 image datasets: MNIST_1, FMNIST_1, $\{\text{EMNIST}_i\}_{i=1}^{3}$, $\{\text{KMNIST}_i\}_{i=1}^{4}$. For each of these image datasets, we set aside 4800 validation images from the training dataset. We also keep the provided test images separate.

Second, we define a set of binary classification tasks, which map $\mathbb{R}^{16} \rightarrow \{0, 1\}$. Each task's labelling function g_i receives two concatenated 8-dimensional one-hot encodings and returns a binary value, indicating if the given combination of 2 classes, represented by the input, fulfils a certain criteria. We further decompose the labelling function into $g_i(\mathbf{x}) = g_k^{(2)}(g_j^{(1)}(\mathbf{x}[:8]), g_j^{(1)}(\mathbf{x}[8:]))$. Here, $g_j^{(1)}$ maps a one-hot encoding to an integer between 1 and 8. For instance, $g_1^{(1)}$ maps the first dimension to 1, the second to 2 and so on. As a result, we use $g^{(1)}$ to convert the initial input of two

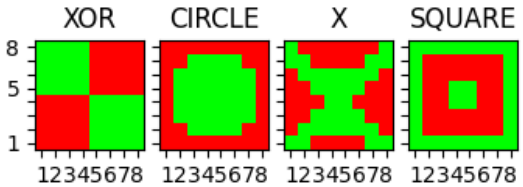


Figure 2: An illustration of the four two-dimensional patterns which are used by the four $g^{(2)}$ functions to label the input coordinates. Green indicates a positive label, and red indicates a negative label.

one-hot encodings to two-dimensional coordinates. We define 4 different $g^{(1)}$ mappings, where $g_1^{(1)}$ is defined as above, and $g_1^{(2)}$, $g_1^{(3)}$ and $g_1^{(4)}$ each map the dimensions to a different randomly selected integer between 1 and 8.

At the same time, each $g_k^{(2)} : \mathbb{R}^2 \to \{0, 1\}$ outputs whether a given two-dimensional coordinate is a part of a certain pattern or not. We define 4 different $g^{(2)}$ functions, each corresponding to one of 4 two-dimensional patterns, shown in Fig 2. In total, these functions need to label $8 * 8 = 64$ different two-dimensional coordinates.

We fuse the 4 different $g^{(1)}$ functions with the 4 different $g^{(2)}$ functions to define 16 different g functions:

$$\{g_{(k-1)*4+j}(\mathbf{x}) = g_k^{(2)}(g_j^{(1)}(\mathbf{x}[:8]), g_j^{(1)}(\mathbf{x}[8:])), k \in \{1, 2, 3, 4\}, j \in \{1, 2, 3, 4\}\}.$$

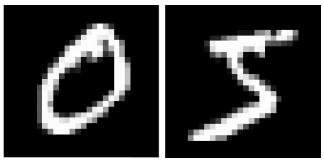


Figure 3: An example input for $\Psi = (D_{\text{MNIST}_1}, h_{\text{MNIST}_1}, g = (g_{\text{XOR}}^{(2)}, g_1^{(1)}))$. These images are classified by h_{MNIST} and then are mapped to the coordinates (1, 6) by $g_1^{(1)}$ since they represent the first and sixth classes respectively. Afterwards, $g_{\text{XOR}}^{(2)}$ labels this input as 0, using the XOR pattern, shown in 2.

Finally, we can combine our 9 image classification datasets $\{(D_i, h_i)\}_{i=1}^{9}$ with our 16 binary classification tasks, in order to create 144 compositional problems $\{\Psi_{(k-1)*9+j} = (D_k, h_k, g_j), k \in \{1, ..., 9\}, j \in \{1, ..., 16\}\}$. The input to a problem $\Psi_i = (D_k, h_k, g_j)$ are two images sampled from D_i. Each image is labelled by h_i, each resulting in an eight-dimensional one-hot encoding of the corresponding image's class. The two one-hot encodings are then concatenated and labelled by g_j, which results in a binary label. An example for $\Psi = (D_{\text{MNIST}_1}, h_{\text{MNIST}_1}, g = (g_{\text{XOR}}^{(2)}, g_1^{(1)}))$ is shown in Fig 3.

Since the sequence S^{sp} involves transferring across input spaces, we define a single additional input domain and an associated classification task $(D_{\text{Ab}}, h_{\text{Ab}})$, $h_{\text{Ab}} : \mathbb{R}^{10} \to \mathbb{R}^8$. We use the Abalone dataset (mis, 1995), in which one needs to predicts the number of rings in a shell of a sea snail, based on a ten-dimensional input vector, containing different characteristics of the snail. The number of rings is an integer, so we represent its prediction as a classification task. We select the 8 classes represented by the highest number of data points, namely $\{6, 7, ..., 13\}$. The resulting classification task can be combined with an upper labelling function in order to constitute a compositional problem, e.g. $\Psi = (D_{\text{Ab}}, h_{\text{Ab}}, g = (g_{\text{XOR}}^{(2)}, g_1^{(1)}))$

B.2 Realising the sequences

To implement a sequence S of length l, we need to select l concrete compositional problems which fit the pattern specified by said sequence. Let the sequence have $l^{(1)}$ different pairs of image domain

and lower labelling sub-function, and $l^{(2)}$ different upper labelling sub-functions. For all sequences, apart from S^{long}, we select $l^{(1)}$ pairs of (D_i, h_i) by sampling from the set of all possible image classification tasks, without replacement. Similarly, we select $l^{(2)}$ different upper labelling sub-functions by sampling without replacement from the set of available binary classification tasks $\{g_i\}_{i=1}^{16}$. For S^{long}, we use sampling with replacement.

If a problem's training dataset needs to be large, Ψ_i^+, we generate it according to the triple $n_{\text{tr}}^+ = (30000, \text{All}_{\text{tr}}, \text{All})$. The first value indicates that we generate 30000 data points in total. The second value indicates how many unique images from the ones set aside for training, are used when generating the inputs. In this case, we use all the available training images. The third value indicates how many out of the 64 unique two-dimensional coordinates, used by the upper labelling sub-function, are represented by the input images. In this case, we use all two-dimensional coordinates.

Some of the problems' training datasets are required to be small and to necessitate transfer. For sequences $S^-, S^{\text{out}}, S^{\text{out*}}, S^{\text{out**}}, S^{\text{few}}, S^+$, we generate the training datasets of each problem Ψ^- using the triple $n_{\text{tr}}^- = (10000, 100, \text{All})$. This way, only 100 unique images are used to generate the training dataset, so solving the problem is likely to be difficult without perceptual transfer. The subset of unique images is randomly sampled and can is different between two problems which share an input domain. For sequences S^{in} and S^{sp}, which evaluate non-perceptual transfer, we use the triple $n_{\text{tr}}^- = (10000, \text{All}_{\text{tr}}, 30)$. As a result, the generated datasets will only represent $30/64$ of the two-dimensional coordinates, which is not sufficient for learning the underlying two-dimensional pattern. Therefore, these problems will necessitate non-perceptual transfer. When generating a dataset for a problem Ψ^- in the sequence S^{long}, we randomly choose between the two, namely between $(10000, 100, \text{All})$ and $(10000, \text{All}_{\text{tr}}, 30)$.

For the problems in which the training dataset needs to contain only a few data points, Ψ^{--}, we use the triple $n_{\text{tr}}^{--} = (10, 20, 10)$. This creates only 10 data points, representing 20 different images and 10 different two-dimensional patterns.

For problems with Ψ^{--}, we use the triple $n_{\text{val}}^{--} = (10, 20, 10)$ for generating the validation dataset. For the rest of the problems, we use the triple $n_{\text{val}}^{--} = (5000, \text{All}_{\text{val}}, \text{All})$. Finally, we generate all test datasets using the triple $n_{\text{test}}^{--} = (5000, \text{All}_{\text{test}}, \text{All})$.

C EXPERIMENTS

Our experiments are designed to evaluate a number of aspects of our work. First, we would like to verify the ability of our benchmark suite, BELL, to diagnose different properties of an LML algorithm. Second, we would like to verify that our LML algorithms, PeCL, and are capable of perceptual, non-perceptual and few-shot transfer and to assess their ability to scale to a long sequence of problems. Third, we would like to assess PeCL's applicability to different image domains, which would evaluate whether our approximations of the modules' projected input distributions are useful across problems sequences with different input domains. Fourth, we would like to examine the difference made by using Bayesian optimisation inside . Finally, we would like to evaluate our algorithms' applicability to different neural architectures and training settings.

To do this, we perform experiments on our benchmark suite, BELL, involving our three LML algorithms and competitive baselines. Moreover, we evaluate our algorithms on the CTrL benchmark suite, presented in Veniat et al. (2020). Finally, we run ablation studies in order to examine some of our design choices.

We compare our algorithms to a number of competitive modular LML baselines. First, we evaluate MOLL-RS , which is a modular LML algorithm, as defined in Algorithm **??**, with a search strategy which suggests paths at random. Random search has been shown to be a competitive baseline in high dimensional search spaces (Wang et al., 2013) and in neural architecture search (Li & Talwalkar, 2020). Second, we evaluate , which is a modular LML algorithm with an exhaustive search strategy. However, simultaneously searches through neural architectures and paths. To keep the results comparable, we fix the neural architecture to the one provided to all other algorithms. Third, we evaluate MNTDP-D (Veniat et al., 2020) which is an efficient modular LML algorithm, capable of perceptual transfer. The algorithm evaluates a constant number L of different paths (where L

is the number of module layers) and is described in detail in the background chapter, section **??**. MTNDP-D has been shown to outperform regularisation LML methods (Kirkpatrick et al., 2017) and repetition-based methods (Chaudhry et al., 2019). Finally, we also use the standalone baseline (SA), which trains a new model on each problem to demonstrate the performances which can be achieved without knowledge transfer. When training, we impose a computational limit on all benchmarks of evaluating at most $r = (2L + t - l_{\text{min}})$ paths. However, MOLL-RS is likely to select many paths which only consist of pre-trained modules and can be quickly evaluated. Therefore, we instead limit its runtime to the time it would take to train r different paths.

Across both benchmark suites, we use the same set of hyperparameters for our algorithms, which we initially selected for a different set of problems in our preliminary experiments. This suggests that this choice of hyperparameters is robust and applicable to different problems and neural architectures. For PeCLand we project the hidden states to 20 dimensions before modelling the resulting distribution. Moreover, we use 0.001 as the softmax temperature parameter for the prior distribution over pre-trained modules. For and we store 40 of the training inputs of each of the pre-trained modules used in the $(L - l_{\text{min}} + 1)$th layer. The value for l_{min} is 3 for both the BELLand the CTrL benchmarks. The jitter of the expected improvement is 0.001 and the threshold used for early stopping, is 0.001. To calculate the value of the lower confidence bound (LCB), we set the hyperparameter $\beta = 2$ in order to encourage exploration over exploitation.

We use two measurements, defined in **??**, to assess the performance of each LML algorithm. First, we compute the average accuracy, $\mathcal{A}$, across all problems after the last problem is solved. Second, we compute the forward transfer on the last problem only, $\mathcal{T}^{-1}$, for the sequences in which the performance on the last problem diagnoses an LML property. We do not evaluate backward transfer, nor forgetting, since all the LML algorithms which we evaluate are immune to forgetting, and to backward transfer, by design as they freeze any previously trained parameters.

In our experiments, we make the training process deterministic, so that the difference in performance can be accredited only to the LML algorithm, and not due to randomness introduced during training. For this purpose, we fix the random initialisation of new parameters to be problem and path-specific. In other words, for a given problem, if a model with the same path is instantiated twice, it will have the same initial values for its new randomly initialised parameters. Moreover, we fix the sequence of randomly selected mini batches seen during training to be the same for a given problem. Finally, as we use PyTorch for our experiments, we fix the random seed and use the command "torch.use_deterministic_algorithms(True)". The overall results is that, for a given problem and a given library, evaluating the same path will always result in the same performance, even across different modular LML algorithms.

All experiments are implemented using PyTorch $1.11.0$ (Paszke et al., 2019). We also use GPy's (GPy, since 2012) implementation of a Gaussian process. We run each LML algorithm on a single sequence, on a separate GPU. All experiments are run on a single machine with two Tesla P100 GPUs with 16 GB VRAM, 64-core CPU of the following model: "Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz", and 377 GB RAM.

C.1 BELLBENCHMARKS

We first evaluate our algorithms and the baselines on the benchmark suite which we introduced in Section B. We create 3 versions of each sequence by randomly selecting different compositional problems. Then, for each sequence, we report the measurements, averaged over these 3 versions.

C.1.1 NEURAL ARCHITECTURE

Here, we present the minimal neural architecture which we have found to be suitable for solving said problems.

We first define a convolutional neural network $\zeta_{\text{CNN}} : \mathbb{R}^{28\times 28} \to \mathbb{R}^8$, suitable for processing images from the image classification datasets. We use a 5-layer architecture with *ReLU* hidden activations and a *softmax* output activation. The layers are as follows: *Conv2d(input_channels=1, output_channels=64, kernel_size=5, stride=2, padding=0)*, *Conv2d(input_channels=64, output_channels=64, kernel_size=5, stride=2, padding=0)*, *flatten*, *FC(4*4*64, 64)*, *FC(64, 64)*,

FC(64, 10). Here, *Conv2d* specified a two-dimensional convolutional layer and *FC* specifies a fully-connected layer.

Second, we define a fully-connected neural network for processing a concatenation of two 8-dimensional one-hot embeddings, $\zeta_{\text{MLP}} : \mathbb{R}^{16} \to \mathbb{R}^{1}$. It consists of 2 *FC* hidden layers with 64 hidden units and *RELU* hidden activations, followed by an output *FC* layer with a $sigmoid$ activation.

For a compositional problem $\Psi_k = (D_i, h_i, g_j)$ the input is a 2-tuple of images, $(\mathbf{x}^1, \mathbf{x}^2)$ and the expected output is a binary classification. We solve it using the architecture $\zeta_{\text{comp}} = \zeta_{\text{MLP}}(\textit{concatenate}(\zeta_{\text{CNN}}(\mathbf{x}^2), \zeta_{\text{CNN}}(\mathbf{x}^2)))$. This architecture processes each of the 2 input images with the same ζ_{CNN} model. Then the 2 outputs are concatenated and processed by a ζ_{MLP} model.

We represent this as a modular neural architecture by considering each of the 8 parameterised non-linear transformations to be a separate module. This increases the number of possible paths for each problem. As a result, for the 6th problem in a sequence, the number of possible paths is upper bounded by $\mathcal{O}(6^8 = 1679616)$. Therefore, in this setting, even sequences of length 6 are challenging for modular LML approaches.

The input space of the last problem of sequence S^{sp}' is given by a ten-dimensional vector. Therefore, only for this problem, we replace ζ_{CNN} with a different achitecture, ζ_{FC}, which consists of two fully connected layers, with a hidden size of 64, uses $ReLU$ as a hidden activation and *softmax* as its output activation.

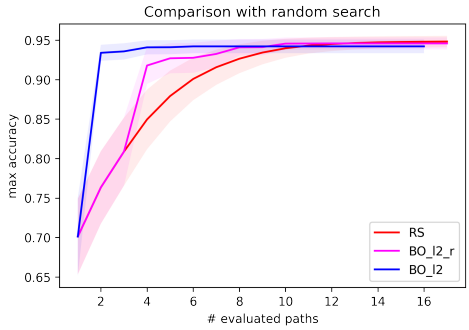
We train new parameters to increase the log likelihood of the labels using the AdamW optimiser (Loshchilov & Hutter, 2017) with 0.00016 learning rate, and 0.97 weight decay. The training is done with a mini batch size of 32 and across 1200 epochs. We apply early stopping, based on the validation loss. We stop after 6000 updates without improvement and return the parameters which were logged to have had the best validation accuracy during training.

C.2 CTrL BENCHMARKS

The CTrL benchmark suite was introduced in Veniat et al. (2020). They define a number of sequences, based on seven image classfication tasks, namely: CIFAR10 and CIFAR100 (Krizhevsky et al., 2009), DTD (Cimpoi et al., 2014), SVHN (Netzer et al., 2011), MNIST (LeCun et al., 1998), RainbowMNIST (Finn et al., 2019), and Fashion MNIST (Xiao et al., 2017). All images are rescaled to 32x32 pixels in the RGB color format. CTrL was first to introduce the following sequences: S^{-}, S^{+}, S^{in}, S^{out}, S^{pl} and S^{long}, which are defined similarly to our definitions. However, the difference is that they are defined for and implemented by image classification tasks. The last task in S^{in}, which evaluates non-perceptual transfer, is given by MNIST images with a different background color than the first task. The last task in S^{out} is given by shuffling the output labels of the first task. S^{long} has 100 tasks. For each task, they sample a random image dataset and a random subset of 5 classes to classify. The number of training data points is sampled according to a distribution that makes it more likely for later tasks to have small training datasets. In contrast to us, they use only 1 selection of tasks for each sequence, i.e. 1 realisation of each sequence. To generate the sequences, we use the code provided by the authors (Veniat & Ranzato, 2021).

The neural architecture used is a small variant of ResNet18 architecture which is divided into 6 modules, each representing a different ResNet block (He et al., 2016). While the paper presenting the CTrL benchmark states that 7 modules are used, we used the authors' code (Veniat, 2021) for this method which specifies only 6 modules with the same total number of parameters. The difference from the architecture stated in the paper is that the output layer is placed in the last module, instead of in a separate module.

All parameters are trained to reduce the cross-entropy loss with an Adam optimiser Kingma & Ba (2014) with $\beta_1 = 0.9$, $\beta_1 = 0.999$ and $\epsilon = 10^{-8}$. For each task, each path is evaluated 6 times with different combinations of values for the hyperparameters of the learning rate ($\{10^{-2}, 10^{-3}\}$) and of the weight decay strength $\{0, 10^{-5}, 10^{-4}\}$. The hyperameters which lead to the best validation performance are selected. Early stopping is employed during training. If no improvement is achieved in 300 training iterations, the parameters with the the best logged validation performance are selected. Data augmentation is also used during training, namely random crops (4 pixels padding and 32x32 crops) and random horizontal reflection.



(a) Comparing to randomly selected initial points (BO_l2_rnd_init), and random search (RS).

D ABLATION EXPERIMENTS

D.0.1 BAYESIAN OPTIMISATION

We evaluate our Bayesian optimisation algorithm $\mathbb{S}_{\text{BO}}^{\text{NT},l}$ which is used by for non-perceptual transfer. For this purpose, we create a new sequence:

$$S^{\text{in}+} = [\Psi_1^+, \Psi_2^+, ..., \Psi_{15}^+, \Psi_{16}^- = (D_6, h_6, g_1)]$$

which involves all 16 upper labelling sub-functions g of BELL. The last problem's dataset is generated according to the triple $n_{\text{tr}}^- = (10000, \text{All}_{\text{tr}}, 30)$, which states that only 30 out of the 64 possible two-dimensional patterns are represented in the dataset. As a result, non-perceptual transfer is necessary in order to maximise the performance on the final problem. We create 5 realisations of $S^{\text{in}+}$ with different randomly selected problems.

Even though our method is deterministic, the baselines we compare it to involve randomly selecting 2 or all random paths. For each such baseline, and each of the 5 realisations of the sequence $S^{\text{in}+}$, we run the baseline 10 times with 10 different random seeds. This results in $10 * 5 = 50$ evaluations per method which we average over when reporting its performance. For each method, we plot its maximum accuracy achieved per number of programs evaluated (Fig **??**), and we also report the average number of programs it evaluates before early stopping (#paths) as well as the maximum accuracy it achieves at the end of its run ($\mathcal{A}$), as shown in Table **??**.

We begin by comparing our algorithm $s_{\text{BO}}^{\text{NT},l}$ (which we denote BO_l2) to an augmented version which randomly selects the initial 2 paths (BO_l2_r), and to a random search baseline which recommends paths in a random order (RS). The results are shown in Fig. 4a. They demonstrate our approach's superior anytime performance.