# A General Framework for Continual Learning of Compositional Structures

**Jorge A. Mendez** [1] **Eric Eaton** [1]

## Abstract

A hallmark of human intelligence is the ability to construct self-contained chunks of knowledge and reuse them in novel combinations for solving different problems. To date, research into compositional learning has largely proceeded separately from work on continual learning. We present a general-purpose framework for continual learning of compositional structures, which separates the learning process into two broad stages: learning how to combine existing components to assimilate a novel problem, and learning how to adapt existing components to accommodate the new problem. This separation trades off the stability required to remember how to solve earlier tasks and the flexibility required to solve new tasks.

## 1. Introduction

A major goal of artificial intelligence is to create an agent capable of acquiring a general understanding of the world. Such an agent would require the ability to continually accumulate and build upon its knowledge. Continual learning addresses this setting, whereby an agent faces a sequence of problems and must strive to capture the knowledge necessary for solving each new task it encounters. If the agent accumulates knowledge in some form of compositional representation, it could then selectively reuse and combine relevant pieces of knowledge to construct novel solutions.

We investigate *how to learn compositional structures in a continual learning setting*, and propose a framework agnostic to the specific algorithms used for learning and the form of the structures being learned. Evoking Piaget's (Piaget, 1976) assimilation and accommodation stages of intellectual development, this framework embodies the benefits of dividing the continual learning process into two stages: striving to solve a new task by combining existing components, and using discoveries from the new task to improve existing components and to construct fresh components if necessary.

Our framework can incorporate various forms of compositional structures and different mechanisms for avoiding catastrophic forgetting. As examples, it can incorporate naïve fine-tuning, experience replay, and elastic weight consolidation (Kirkpatrick et al., 2017) as knowledge retention mechanisms, and linear combinations of linear models (Ruvolo & Eaton, 2013) and soft layer ordering (Meyerson & Miikkulainen, 2018) as the compositional structures. We instantiate our framework with these examples, and evaluate it on eight data sets, showing that separating the continual learning process into two stages increases the capabilities of the learning system, reducing catastrophic forgetting and achieving higher overall performance.

## 2. Related Work

In continual learning, agents must learn multiple tasks sequentially, and should accumulate knowledge so as to more efficiently learn to solve new problems. Recent efforts have mainly focused on avoiding catastrophic forgetting. Different algorithms define parts of models to share across tasks. As the agent encounters tasks, it strives to retain the knowledge that enabled it to solve earlier tasks. One common approach is to impose regularization to prevent parameters from deviating in directions that would harm performance on the early tasks (Kirkpatrick et al., 2017; Zenke et al., 2017; Li & Hoiem, 2017; Ritter et al., 2018). Another approach retains a buffer of data from all tasks, and updates the model parameters with data from all tasks, thereby maintaining the knowledge required to solve them (Lopez-Paz & Ranzato, 2017; Nguyen et al., 2018; Isele & Cosgun, 2018).

These approaches avoid catastrophic forgetting, but do not seek to discover reusable knowledge. Although model parameters are reused across tasks, it is unclear what this reusability means, and the way parameters are reused is hard-coded into the architecture design. This latter issue is a major drawback when learning highly varied tasks, as the way in which tasks are connected is often unknown. Algorithms should determine these connections autonomously.

[1]Department of Computer and Information Science, University of Pennylsvania. Correspondence to: Jorge A. Mendez <mendezme@seas.upenn.edu>.

*Continual Learning Workshop at the International Conference on Machine Learning (ICML), 2020.*

Other approaches learn factored or clustered models (Ruvolo & Eaton, 2013; Nagabandi et al., 2019). These algorithms learn a set of models that are reusable across tasks and automatically select how to reuse them. However, such methods selectively reuse entire models, enabling knowledge reuse, but not explicitly in a compositional manner.

A mostly distinct line of work has explored the learning of compositional knowledge. The majority of such methods either learn the structure for piecing together a given set of components (Xu et al., 2018; Bunel et al., 2018; Cai et al., 2017) or learn the set of components given a known structure for how to compose them (Bošnjak et al., 2017).

When neither the structure nor the set of components are given and the agent must autonomously discover both, some approaches assume access to a solution descriptor (e.g., in natural language), which the agent can map to a solution structure (Hu et al., 2017; Johnson et al., 2017; Pahuja et al., 2019). Other approaches do not make such an assumption, and instead learn directly from optimization of a cost function (Rosenbaum et al., 2018; Kirsch et al., 2018; Meyerson & Miikkulainen, 2018; Alet et al., 2018; Chang et al., 2019).

Approaches above assume access to a large batch of tasks, making it possible to evaluate numerous combinations of components and structures on all tasks simultaneously. More realistically, the agent will face tasks in a continual learning fashion. Most work in this line assumes that each component can be fully learned by training on a single task, and then can be reused for other tasks (Reed & de Freitas, 2016; Fernando et al., 2017; Valkov et al., 2018). Unfortunately, this is infeasible if the agent has access to little data for each task. One exception was proposed by Gaunt et al. (2017), which improves early components with experience in new tasks, but is limited to very simplistic settings.

In contrast, we continually learn compositional structures without a large batch of tasks or the ability to learn definitive components after training on a single task. Instead, we train on a small initial batch of tasks, and then autonomously update the existing components to accommodate new tasks.

## 3. The Continual Learning Problem

We frame continual learning as online multi-task learning. The agent (Figure 1) will face a sequence of tasks over its lifetime. Each task will be a learning problem defined by a cost function $\mathcal{L}^{(t)}(f^{(t)})$, where the agent must learn a prediction function $f^{(t)} \in \mathcal{F} : \mathcal{X}^{(t)} \mapsto \mathcal{Y}^{(t)}$ to minimize the cost. The solution to each task is parameterized by $\boldsymbol{\theta}^{(t)}$, such that $f^{(t)} = f_{\boldsymbol{\theta}^{(t)}}$. The goal of the continual learner is to find the set of parameters $\{\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(T_{\text{max}})}\}$ that minimizes the cost across all tasks: $\frac{1}{T_{\text{max}}} \sum_{t=1}^{T_{\text{max}}} \mathcal{L}^{(t)}(f^{(t)})$. The total number of tasks, the order in which these tasks will arrive, and the task relationships are all unknown.

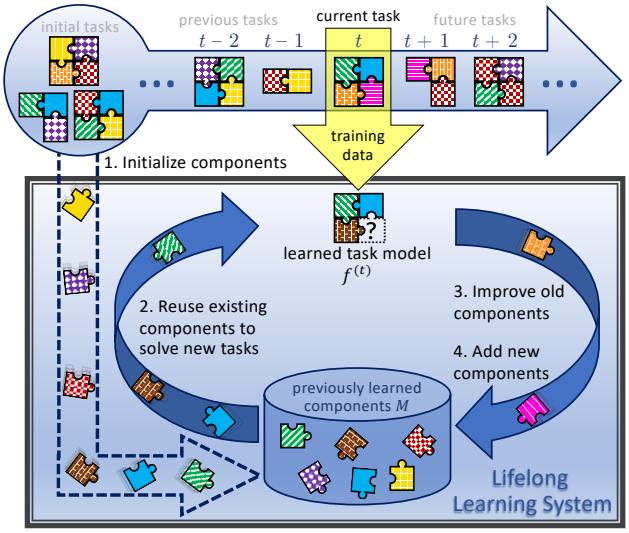


Figure 1: In continual compositional learning, a set of components are initialized from a small set of tasks (1). Each new task is learned by composing the relevant pieces of knowledge (2), improving imperfect components (3), and adding any new components that were discovered (4).

The agent will have access to limited data for each task, and will strive to discover any relevant information to 1) relate it to previously stored knowledge in order to permit transfer and 2) store any new knowledge for future reuse. At any time, the agent may be evaluated on any previous task, requiring the agent to perform well on *all* tasks, so it must strive to retain knowledge from even the earliest tasks.

## 4. The Continual Compositional Learning Framework

Our framework for continual learning of compositional structures will store knowledge in a set of shared components $M = \{m_1, \ldots, m_k\}$. Each component $m_i \in \mathcal{M}$ is a self-contained, reusable function parameterized by $\boldsymbol{\phi}_i$ ($m_i = m_{\boldsymbol{\phi}_i}$) that can be combined with other components. Given these components, the agent will reconstruct each task's predictive function via a task-specific structure $s^{(t)} : \mathcal{X}^{(t)} \times \mathcal{M}^k \mapsto \mathcal{F}$, such that $f^{(t)}(\boldsymbol{x}) = s^{(t)}(\boldsymbol{x}, M)(\boldsymbol{x})$, where $s^{(t)}$ is parameterized by a vector $\boldsymbol{\psi}^{(t)}$. The structure functions select components from $M$ to execute for the current task and input, and the order in which to execute them. Concrete examples are described in Section 4.1.

Intuitively, at any point in the lifetime of an agent, it will have acquired a strong set of components. If these components, with minor adaptations, can be combined to solve the current task, the agent should first learn how to reuse them before making any modifications to them: modifications in the early stages of training, before acquiring sufficient knowledge about the current task, could be damaging to

existing components. Once the structure has been learned, we consider that the agent has captured sufficient knowledge about the current task, and it would be sensible to update the components to better accommodate it. If it is not possible to capture the current task with the existing components, then new components should be added. Our framework is split into the following four steps.

**Initialization** Components should be initialized to be reusable, both across tasks and across "positions" within a task. The latter means, for example, that the modules in deep nets could be used at different depths. One way to achieve this is to enforce a random structure for an initial set of tasks that reuses components both at different positions and across different tasks, and train these initial tasks jointly.

**Assimilation** Approaches to finding compositional structures treat component selection as a reinforcement learning problem, learn it via stochastic search, or define it as a differentiable gating network or softmax selector and train it jointly with the components via backpropagation. Our framework will use any of these approaches to assimilate the current task by keeping the components $M$ fixed and learning only the structure $s^{(t)}$, decoupling the learning of the structure from the learning of the components themselves.

**Accommodation (1)** The first step of accommodation is incorporating newly discovered knowledge into existing components, simultaneously avoiding catastrophic forgetting and incorporating new knowledge about the current task. In non-compositional learning, approaches fine-tune models on the current task, impose regularization to freeze weights, or use experience replay to avoid forgetting. We will instantiate our framework by using any of these methods to accommodate new knowledge into existing components once the current task has been assimilated.

**Accommodation (2)** In the second step of accommodation, the learner must incorporate novel components that encode distinct knowledge than that already available. Autonomously discovering new components adds the flexibility required to handle a lifetime of learning. One way to achieve this is to train the agent with some additional components, and choose to keep these components only if they substantially benefit the agent's performance on the current task.

### 4.1. Compositional Structures

We now present two different instantiations of our framework with varying compositional structures.

**Linear combinations of models** In the simplest setting, each component is a linear model, composed via linear combinations. Specifically, we assume that $\mathcal{X}^{(t)} \subseteq \mathbb{R}^d$, and each task-specific function is given by $f_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{x}) = {\boldsymbol{\theta}^{(t)}}^\top \boldsymbol{x}$, with $\boldsymbol{\theta}^{(t)} \in \mathbb{R}^d$. The predictive functions are constructed from a set of linear component functions: $m_{\boldsymbol{\phi}_i}(\boldsymbol{x}) = {\boldsymbol{\phi}_i}^\top \boldsymbol{x}$ with $\boldsymbol{\phi}_i \in \mathbb{R}^d$, by linearly combining them via a task-specific weight vector: $f^{(t)}(\boldsymbol{x}) = s_{\boldsymbol{\psi}^{(t)}}(\boldsymbol{x}, M)(\boldsymbol{x}) = {\boldsymbol{\psi}^{(t)}}^\top (\boldsymbol{\Phi}^\top \boldsymbol{x})$ where $\boldsymbol{\psi}^{(t)} \in \mathbb{R}^k$ and we have constructed the matrix $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_k]$ to collect all $k$ components.

**Soft layer ordering** To handle more complex models, we construct modular deep nets that compute each layer's output as a linear combination of the outputs of multiple modules. We assume that each module is one layer, the number of components matches the network's depth, and all components share the input and output dimensions (Meyerson & Miikkulainen, 2018). Concretely, each component is a layer $m_{\boldsymbol{\phi}_i}(\boldsymbol{x}) = \sigma({\boldsymbol{\phi}_i}^\top \boldsymbol{x})$, where $\sigma$ is any activation and $\boldsymbol{\phi}_i \in \mathbb{R}^{\tilde{d} \times \tilde{d}}$. Given input $(\mathcal{E}^{(t)} : \mathcal{X}^{(t)} \mapsto \mathbb{R}^{\tilde{d}})$ and output $(\mathcal{D}^{(t)} : \mathbb{R}^{\tilde{d}} \mapsto \mathcal{Y}^{(t)})$ transformations, a matrix $\boldsymbol{\psi}^{(t)} \in \mathbb{R}^{k \times k}$ weights the output of the components at each depth $s^{(t)} = \mathcal{D}^{(t)} \circ \sum_{i=1}^k \boldsymbol{\psi}_{i,1}^{(t)} m_i \circ \cdots \circ \sum_{i=1}^k \boldsymbol{\psi}_{i,k}^{(t)} m_i \circ \mathcal{E}^{(t)}$. The weights are restricted to sum to one at each depth.

## 5. Experimental Evaluation

**Instantiations** We evaluated our framework with the architectures of Section 4.1, varying the method for accommodation (1). **VAN** updates components via backpropagation with data for the current task, ignoring past tasks. **EWC** quadratically penalizes deviations from the previous tasks' parameters and carries out backpropagation on the penalized objective. **ER** takes backpropagation steps with data from a replay buffer along with the current task's data. All algorithms assimilate the current task $t$ via backpropagation on the structure's parameters $\boldsymbol{\psi}^{(t)}$. For soft layer ordering, accommodation (2) considers adding one new component for each task, keeping it only if it yields a substantial improvement in validation accuracy. Assimilation and accommodation (1) alternate backpropagation steps with and without the new component. Intermittently bypassing the new component ensures that, if the agent discards it, the structure over only existing components is optimal. We denote algorithms with and without accommodation (2) as **dynamic + compositional** and **compositional**, respectively.

**Baselines** For each accommodation (1) method, 1) **joint** baselines use compositional structures, but train components and structures jointly, while **no-components** (i.e., vanilla) baselines optimize a single architecture for all tasks. We also trained an ablated version of our framework that keeps all components fixed after initialization (**FM**), only taking assimilation steps for each new task.

See App. A for details of data sets and empirical settings.

### 5.1. Linear combinations of models

Table 1 summarizes results with linear models. Compositional variants clearly outperformed joint versions, sug-

Table 1: Average final performance across tasks using factored linear models—accuracy for FERA and Landmine (higher is better) and RMSE for Schools (lower is better). Standard errors after $\pm$.

| Base | Algorithm | FERA | Landmine | Schools |
|---|---|---|---|---|
| ER | Compositional | $\mathbf{79.0 \pm 0.4}\%$ | $\mathbf{93.6 \pm 0.1}\%$ | $10.65 \pm 0.04$ |
| | Joint | $78.2 \pm 0.4\%$ | $90.5 \pm 0.3\%$ | $11.55 \pm 0.09$ |
| | No Comp. | $66.4 \pm 0.3\%$ | $93.5 \pm 0.1\%$ | $\mathbf{10.34 \pm 0.02}$ |
| EWC | Compositional | $\mathbf{79.0 \pm 0.4}\%$ | $\mathbf{93.7 \pm 0.1}\%$ | $\mathbf{10.55 \pm 0.03}$ |
| | Joint | $67.9 \pm 0.6\%$ | $73.0 \pm 2.4\%$ | $23.32 \pm 1.99$ |
| | No Comp. | $57.0 \pm 0.9\%$ | $92.7 \pm 0.4\%$ | $18.02 \pm 1.04$ |
| VAN | Compositional | $\mathbf{79.0 \pm 0.4}\%$ | $\mathbf{93.7 \pm 0.1}\%$ | $\mathbf{10.87 \pm 0.07}$ |
| | Joint | $67.9 \pm 0.6\%$ | $72.8 \pm 2.5\%$ | $25.80 \pm 2.35$ |
| | No Comp. | $57.0 \pm 0.9\%$ | $92.7 \pm 0.4\%$ | $18.01 \pm 1.04$ |

Table 2: Average final accuracy across tasks using soft layer ordering. Standard errors after $\pm$.

| Base | Algorithm | MNIST | Fashion | CUB | CIFAR | Omniglot |
|---|---|---|---|---|---|---|
| ER | Dyn. + Comp. | $\mathbf{97.7 \pm 0.2}\%$ | $\mathbf{96.3 \pm 0.4}\%$ | $77.6 \pm 0.8\%$ | $\mathbf{75.9 \pm 0.5}\%$ | $\mathbf{69.0 \pm 0.7}\%$ |
| | Compositional | $96.5 \pm 0.2\%$ | $95.3 \pm 0.7\%$ | $\mathbf{79.2 \pm 0.7}\%$ | $56.0 \pm 0.8\%$ | $67.8 \pm 1.0\%$ |
| | Joint | $94.2 \pm 0.3\%$ | $94.7 \pm 0.7\%$ | $76.8 \pm 0.5\%$ | $63.8 \pm 0.6\%$ | $67.9 \pm 0.5\%$ |
| | No Comp. | $91.2 \pm 0.3\%$ | $93.1 \pm 0.6\%$ | $43.1 \pm 1.0\%$ | $49.5 \pm 0.8\%$ | $40.0 \pm 3.9\%$ |
| EWC | Dyn. + Comp. | $\mathbf{97.3 \pm 0.2}\%$ | $\mathbf{96.1 \pm 0.4}\%$ | $\mathbf{72.7 \pm 0.9}\%$ | $\mathbf{71.7 \pm 0.9}\%$ | $\mathbf{67.7 \pm 0.6}\%$ |
| | Compositional | $96.7 \pm 0.2\%$ | $95.3 \pm 0.6\%$ | $72.4 \pm 1.2\%$ | $43.4 \pm 1.1\%$ | $52.2 \pm 7.3\%$ |
| | Joint | $66.3 \pm 1.4\%$ | $69.1 \pm 1.4\%$ | $65.3 \pm 0.7\%$ | $41.9 \pm 0.8\%$ | $61.9 \pm 1.1\%$ |
| | No Comp. | $64.3 \pm 0.8\%$ | $58.5 \pm 2.9\%$ | $47.7 \pm 1.4\%$ | $35.3 \pm 0.7\%$ | $66.2 \pm 1.0\%$ |
| VAN | Dyn. + Comp. | $\mathbf{97.4 \pm 0.3}\%$ | $\mathbf{96.0 \pm 0.4}\%$ | $72.5 \pm 0.8\%$ | $\mathbf{69.6 \pm 1.2}\%$ | $\mathbf{67.1 \pm 0.6}\%$ |
| | Compositional | $96.4 \pm 0.2\%$ | $95.3 \pm 0.6\%$ | $\mathbf{73.7 \pm 1.1}\%$ | $52.5 \pm 1.3\%$ | $65.3 \pm 1.2\%$ |
| | Joint | $67.4 \pm 1.4\%$ | $66.1 \pm 2.4\%$ | $64.4 \pm 0.8\%$ | $41.4 \pm 0.8\%$ | $60.2 \pm 1.1\%$ |
| | No Comp. | $64.4 \pm 1.1\%$ | $59.4 \pm 2.7\%$ | $48.3 \pm 1.9\%$ | $34.1 \pm 0.8\%$ | $64.7 \pm 1.0\%$ |
| FM | Dyn. + Comp. | $\mathbf{99.1 \pm 0.0}\%$ | $\mathbf{97.0 \pm 0.3}\%$ | $78.2 \pm 0.4\%$ | $\mathbf{74.3 \pm 0.9}\%$ | $\mathbf{67.7 \pm 0.7}\%$ |
| | Compositional | $84.1 \pm 0.8\%$ | $85.9 \pm 1.3\%$ | $\mathbf{79.2 \pm 0.6}\%$ | $46.0 \pm 1.6\%$ | $58.3 \pm 3.0\%$ |

gesting that separating the learning into assimilation and accommodation stages enables the agent to better capture the structure of the problem. No-components variations, which learn a single linear model for all tasks, performed better than joint versions in two out of the three data sets, and even outperformed our compositional ER algorithm in one case. This indicates that the tasks in those two data sets are so closely related that a single model can capture them.

### 5.2. Soft layer ordering

We then evaluated how different algorithms performed when learning deep nets with soft layer ordering. Results in Table 2 show that all the algorithms conforming to our framework outperformed the joint and no-components learners. In four out of the five data sets, the dynamic addition of new components yielded either no or marginal improvements. However, in the CIFAR data set, it was crucial for the agent to be capable of detecting when new components were needed. This added flexibility enables our learners to handle more varied tasks, where new problems may not be solved without substantially new knowledge. Algorithms with accommodation (1) outperformed the ablated compositional FM agent, showing that it is necessary to accommodate new knowledge into the set of components in order to handle a diversity of tasks. When FM was allowed to dynamically add new components (keeping old ones fixed), it yielded the best performance on MNIST and Fashion by adding far more components than methods with accommodation (1), as we show in Appendix B.

## 6. Conclusion

We presented a framework for learning compositional structures in a continual learning setting. The key piece of our framework is the separation of the learning into two stages: assimilation of new problems with existing components, and accommodation of newly discovered knowledge into the set of components. We showed the flexibility of our framework instantiating it into six different algorithms, and demonstrated empirically that algorithms within our framework are stronger continual learners than existing approaches.

## Acknowledgements

The research presented in this paper was partially supported by the Lifelong Learning Machines program from DARPA/MTO under grant #FA8750-18-2-0117.

## References

Alet, F., Lozano-Perez, T., and Kaelbling, L. P. Modular meta-learning. In *Proceedings of the 2nd Conference on Robot Learning*, pp. 856–868, 2018.

Bošnjak, M., Rocktäschel, T., Naradowsky, J., and Riedel, S. Programming with a differentiable forth interpreter. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 547–556, 2017.

Bunel, R., Hausknecht, M., Devlin, J., Singh, R., and Kohli, P. Leveraging grammar and reinforcement learning for neural program synthesis. In *6th International Conference on Learning Representations*, 2018.

Cai, J., Shin, R., and Song, D. Making neural programming architectures generalize via recursion. In *5th International Conference on Learning Representations*, 2017.

Chang, M., Gupta, A., Levine, S., and Griffiths, T. L. Automatically composing representation transformations as a means for generalization. In *7th International Conference on Learning Representations*, 2019.

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D. PathNet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

Gaunt, A. L., Brockschmidt, M., Kushman, N., and Tarlow, D. Differentiable programs with neural libraries. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1213–1222, 2017.

Hu, R., Andreas, J., Rohrbach, M., Darrell, T., and Saenko, K. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the 2017 IEEE International Conference on Computer Vision*, pp. 804–813, 2017.

Isele, D. and Cosgun, A. Selective experience replay for lifelong learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pp. 3302–3309, 2018.

Johnson, J., Hariharan, B., van der Maaten, L., Hoffman, J., Fei-Fei, L., Lawrence Zitnick, C., and Girshick, R. Inferring and executing programs for visual reasoning. In *Proceedings of the 2017 IEEE International Conference on Computer Vision*, pp. 2989–2998, 2017.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

Kirsch, L., Kunze, J., and Barber, D. Modular networks: Learning to decompose neural computation. In *Advances in Neural Information Processing Systems 31*, pp. 2408–2418, 2018.

Li, Z. and Hoiem, D. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2017.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems 30*, pp. 6467–6476, 2017.

Meyerson, E. and Miikkulainen, R. Beyond shared hierarchies: Deep multitask learning through soft layer ordering. In *6th International Conference on Learning Representations*, 2018.

Nagabandi, A., Finn, C., and Levine, S. Deep online learning via meta-learning: Continual adaptation for model-based RL. In *7th International Conference on Learning Representations*, 2019.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. Variational continual learning. In *6th International Conference on Learning Representations*, 2018.

Pahuja, V., Fu, J., Chandar, S., and Pal, C. Structure learning for neural module networks. In *Proceedings of the Beyond Vision and LANguage: inTEgrating Real-world kNowledge (LANTERN)*, pp. 1–10. Association for Computational Linguistics, 2019.

Piaget, J. *Piaget's Theory*, pp. 11–23. Springer Berlin Heidelberg, Berlin, Heidelberg, 1976.

Reed, S. and de Freitas, N. Neural programmers-interpreters. In *4th International Conference on Learning Representations*, 2016.

Ritter, H., Botev, A., and Barber, D. Online structured Laplace approximations for overcoming catastrophic forgetting. In *Advances in Neural Information Processing Systems 31*, pp. 3738–3748, 2018.

Rosenbaum, C., Klinger, T., and Riemer, M. Routing networks: Adaptive selection of non-linear functions for multi-task learning. In *6th International Conference on Learning Representations*, 2018.

Ruvolo, P. and Eaton, E. ELLA: An efficient lifelong learning algorithm. In *Proceedings of the 30th International Conference on Machine Learning*, pp. 507–515, 2013.

Valkov, L., Chaudhari, D., Srivastava, A., Sutton, C., and Chaudhuri, S. Houdini: Lifelong learning as program synthesis. In *Advances in Neural Information Processing Systems 31*, pp. 8687–8698, 2018.

Xu, D., Nair, S., Zhu, Y., Gao, J., Garg, A., Fei-Fei, L., and Savarese, S. Neural task programming: Learning to generalize across hierarchical tasks. In *Proceedings of*

*the 2018 IEEE International Conference on Robotics and Automation*, pp. 3795–3802, 2018.

Zenke, F., Poole, B., and Ganguli, S. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 3987–3995, 2017.