



universität
wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

A Study of Meta-Learned Hebbian Plasticity and Its Architectural
Implications

verfasst von | submitted by
Marlon Jacob Youngerman, BSc (Hons)

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of
Master of Science (MSc)

Wien, | Vienna, 2026

Studienkennzahl lt. Studienblatt | degree
programme code as it appears on the student
record sheet:

UA 066 935

Studienrichtung lt. Studienblatt | degree
programme as it appears on the student record
sheet:

Masterstudium Medieninformatik

Betreut von | Supervisor:

Univ.-Prof. Dr. Uwe Zdun

Contents

1	Introduction	11
1.1	Domain randomisation	12
1.2	Meta learning	13
1.3	Hebbian learning	13
1.4	Architectural Paradigms of Adaptation	14
1.5	Research Questions	15
2	Literature Review	17
2.1	Introduction	17
2.2	Background	18
2.2.1	Learning to learn	18
2.2.2	Hebbian Learning and Synaptic Plasticity	19
2.3	Plasticity in Meta-Learning: A Conceptual Overview	21
2.4	Advantages and Limitations	22
2.5	Applications	26
2.5.1	Continual Learning	26
2.5.2	Few-shot Learning	26
2.5.3	Reinforcement Learning	27
2.6	Survey of Existing Work	28
2.7	The Case for Hebbian Plasticity: A Comparison with Other Meta-Learning Approaches	28
3	Proposed Method	31
3.1	Universal Control Across Embodiments	31
3.2	Models Compared	31
3.3	Comparison of Model Complexity and Capacity	33
3.4	Architectural Complexity and Software Engineering Trade-offs	34
3.5	System Architecture for Massive Parallelization	35
3.5.1	Computational Challenges of Meta-Learning	35

3.5.2	Unified GPU-Based Execution Pipeline	35
3.5.3	Accelerator(GPU) Native Components	36
4	Experimental Setup	39
4.1	Description of experiment	39
4.2	Choice of environment	39
4.3	Morphology Generation	41
4.4	Reward Function	42
4.5	Training Procedure	43
5	Experimental Results and Discussion	45
5.1	Training Performance	45
5.2	Generalization to Unseen Morphologies	46
5.3	The Mean Episodic Reward (MER)	48
5.4	Limitations and Threats to Validity	49
6	Software Architecture Aspects	51
6.1	Overview	51
6.2	MLOps	51
6.3	The End-to-End Machine Learning Lifecycle	52
6.4	Architectural Implications of Continuous Adaptation	54
6.5	Rethinking the Foundations of MLOps for Continual Reinforcement Learning	55
7	Conclusion and Future Work	59
7.1	Thesis Summary	59
7.2	Research Question Evaluation	59
7.2.1	RQ1: Does meta-learned plastic networks outperform static networks in adapting to unseen environments?	59
7.2.2	RQ2: Does increasing the expressivity of the local plasticity rule improve generalization?	60
7.2.3	RQ3: How does plasticity alter the MLOps lifecycle compared to traditional external retraining pipelines?	60
7.2.4	RQ4: What architectural patterns and safety mechanisms are required to monitor, validate, and control model behaviour during continuous deploy- ment?	60

7.3	Key Findings	61
7.4	Future Work	62
7.4.1	Experimental Extensions	62
7.4.2	Neuromorphic and Spiking Systems	63
Appendix A: Reproduction of the Original RL² Paper		65
A.1	Abstract	65
A.2	Introduction	65
A.3	Method	66
A.3.1	Reinforcement Learning Formalism	66
A.3.2	Meta-RL	67
A.3.3	Explanation of the RL ² method	67
A.4	Experiments	68
A.4.1	Multi-Armed Bandits	68
A.4.2	Tabular MDPs	68
A.4.3	Maze	69
A.5	Practical Recommendations	70

Danksagung

Mein besonderer Dank gilt Univ.-Prof. Dr. Uwe Zdun für die Möglichkeit, diese Masterarbeit unter seiner Betreuung zu verfassen, sowie für das entgegengebrachte Vertrauen und die Unterstützung während der Arbeit. Außerdem möchte ich mich herzlich bei Zizou Fang für seine hilfreiche Unterstützung, seine Rückmeldungen und die Begleitung während des Projekts bedanken. Ein besonderer Dank gilt auch meinem Kater Thunfisch, der mich während des Schreibprozesses begleitet und mit seiner Anwesenheit für viele kleine Pausen und Motivation gesorgt hat.

Zusammenfassung

Agenten des Reinforcement Learning haben oft Schwierigkeiten, sich an Situationen anzupassen, die sich von den Trainingsbedingungen unterscheiden. Diese Masterarbeit untersucht, ob Hebb'sche Plastizität die Anpassungsfähigkeit und Generalisierung solcher Agenten verbessern kann. Inspiriert von biologischer synaptischer Plastizität ermöglicht Hebb'sches Lernen, dass sich Netzwerkgewichte während einer Episode auf Grundlage lokaler Aktivitätsmuster verändern, anstatt nach dem Training festgelegt zu bleiben.

Die Arbeit konzentriert sich auf eine Bewegungssteuerung, die auf unterschiedliche Morphologien generalisieren soll. Dafür wurden Agenten auf einer kleinen Auswahl von Morphologien trainiert und anschließend auf unbekannten Varianten evaluiert, um ihre Fähigkeit zur Generalisierung über die Trainingsverteilung hinaus zu testen.

Die Ergebnisse zeigen, dass das Hebb'sche Modell schneller lernt und eine höhere Belohnung erreicht. Dies deutet darauf hin, dass meta-gelernte Plastizität Vorteile für schnelle Anpassung bieten kann. Insgesamt zeigt diese Arbeit sowohl das Potenzial als auch die Grenzen Hebb'scher Plastizität als Mechanismus für robustes Reinforcement Learning in anspruchsvollen Umgebungen.

Die Arbeit untersucht außerdem die Auswirkungen kontinuierlich adaptiver Systeme wie des Hebb'schen Lernens auf Softwarearchitektur und MLOps und diskutiert die Herausforderungen für Monitoring, Validierung und Deployment, wenn sich Modellparameter während des Betriebs verändern.

Abstract

Reinforcement learning agents often struggle to adapt to situations that differ from those encountered during training. This thesis investigates whether Hebbian plasticity can improve adaptation and generalization in such settings. Inspired by biological synaptic plasticity, Hebbian learning allows neural network weights to change during an episode based on local activity patterns, rather than remaining fixed after training.

The study focuses on a locomotion control policy capable of generalization to different morphologies. Agents were trained on a small set of morphologies and evaluated on unseen variants to test their ability to generalize beyond the training distribution.

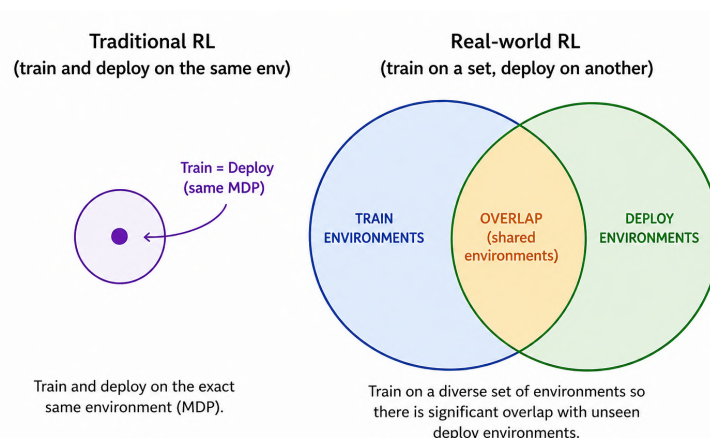
The results show that the Hebbian model learns faster and reaches a higher reward. The findings suggest that meta-learned plasticity can provide benefits for rapid adaptation. Overall, this thesis highlights both the promise and limitations of Hebbian plasticity as a mechanism for robust reinforcement learning in challenging environments.

Additionally, the thesis discusses the software architecture and MLOps implications of continuously adaptive systems such as Hebbian learning, and the challenges of monitoring, validation, and deployment when model parameters change during runtime.

1 Introduction

Traditionally, reinforcement learning (RL) assumes that training and deployment happen in exactly the same environment. However, in real-world application of RL, the deployment environment will differ from the training one. This is because real-world systems are subject to variability, noise, and changing conditions that are difficult to fully capture during training, leading to inevitable distribution shifts between training and deployment. An example of this problem is in sim-to-real settings, where policies trained in simulation often face a gap in dynamics and observations when transferred to the real world. One solution to these issues is to learn across a distribution of environments rather than a single one. The agent is exposed to a diverse set of environments, while deployment occurs on another set of environments that may differ but share underlying structure. The two key ideas are firstly, that by training on diverse environments, we increase the chance that deployment scenarios fall within this overlap. Secondly the agent that successfully adapt to unseen situations are rewarded, enabling robust performance despite inevitable perturbations. Traditional deep learning relies on "frozen" weights after training. This makes them brittle in changing environments. Because the policy doesn't change during deployment. We want agents that can adapt to new tasks or morphologies on the fly.

A way to formalize the distribution over environments is to view it as a single partially observ-



able Markov decision process (POMDP) Humplik et al. (2019). In this interpretation, the agent does not know which environment in the set, it is interacting with at the start of an episode. This uncertainty can be captured by a latent, unobserved variable that encodes the task specification (e.g., reward function or transition dynamics). The agent must infer this hidden context from its interaction history (observations, actions, and rewards) and adapt its behaviour accordingly. This unifies the problem: rather than solving many separate MDPs, the agent is effectively solving one POMDP in which uncertainty over the environment is part of the state.

Therefore, it is formally equivalent to say that we are training on a single partially observable Markov decision process or on a set of MDPs.

For small scale settings, the task inference framing is appropriate, however in most settings it doesn't reflect the full difficulty of the problem. The big world hypothesis Javed and Sutton (2024) posits that, in many real-world learning problems, the environment is vastly more complex than the agent itself. However, this condition however is rarely fulfilled in many papers, the authors give the example of a DQN agent Mnih et al. (2013) on the Atari benchmark Bellemare et al. (2013), where the agent requires orders of magnitude more computation than the environment itself. As a result of being in a "Big World", the agent cannot fully observe, represent, or learn all relevant aspects of the world. Instead, it must operate under severe computational and memory constraints, relying on partial information to achieve its goals. Under this hypothesis the agents can not eventually learn a complete and stable model of their environment. Adaptation can not correspond to inferring a fixed latent task, with the assumption that sufficient experience allows the agent to recover the true underlying environment. Under the big world hypothesis, instead of converging to a correct task identity, the agent must continuously update a partial policy, storing information that is useful in the current context, and discarding information that is no longer useful.

1.1 Domain randomisation

The method of training on a set of environments, was first introduced by Tobin et al. (2017) under the name of domain randomisation (DR), it acts as a form of regularization that forces a reinforcement learning agent to prioritize the acquisition of universal task invariants over environment-specific heuristics. Originally used for its utility in sim-to-real transfer, domain randomization acknowledges the 'Big World' reality by purposefully varying the physical parameters of the environment (e.g. friction, gravity, or morphology) during training. By doing so, we transform the simulation into a distribution of possible worlds. In this way a Big World is created without increasing computation demand, through a collection of Small Worlds. For the agent, this means that a single,

static set of weights is no longer sufficient; the environment's complexity now exceeds the agent's capacity to 'memorize' a solution for every possible permutation.

Another example of domain randomisation is OpenAI et al. (2019), where DR was used to train a robotic hand to solve Rubik's cube. The authors theorize "Our main hypothesis that motivates ADR [Automatic Domain Randomization] is that training on a maximally diverse distribution over environments leads to transfer via emergent meta-learning. More concretely, if the model has some form of memory, it can learn to adjust its behaviour during deployment to improve performance on the current environment over time, i.e. by implementing a learning algorithm internally. We hypothesize that this happens if the training distribution is so large that the model cannot memorize a special-purpose solution per environment due to its finite capacity." In other words, in the underparametrized, "big world" regime, the inability to memorize all possible environments forces the agent to rely on adaptive strategies rather than fixed policies. If the agent is equipped with any self-modifying capability or intra-episodic memory, in this setting, it will naturally learn to efficiently adjust to the current context. This perspective leads to meta-learning, where the objective is not to solve a task, but to acquire priors that facilitate rapid adaptation across tasks.

1.2 Meta learning

Meta-learning provides a natural framework for addressing the challenges imposed by the big world hypothesis. Rather than attempting to learn a fixed policy that performs well across all environments, meta-learning aims to automatically develop distribution-specific learning algorithms capable of rapid, on-the-fly adaptation to new situations. In this framework, the objective shifts from solving individual tasks to learning how to efficiently adjust behaviour based on limited experience. Instead of encoding a fixed solution, the agent learns an adaptation strategy that allows it to adjust its behaviour online as it interacts with a new environment. The essence of meta-learning is the shift from task-specific optimization to distribution-level acquisition of priors, enabling few-shot learning by leveraging past experiences to bypass the need for massive datasets in new environments.

1.3 Hebbian learning

A key question arising from the meta-learning perspective is how the adaptation process should be implemented. One biologically inspired class of update mechanisms is Hebbian learning, where weight changes are driven by local interactions between neural activations. Unlike standard neural networks with fixed weights during deployment, Hebbian plasticity allows the agent to continu-

ously modify its parameters based on its ongoing experience. Previous work, such as Najarro and Risi (2020), has shown that embedding plasticity into neural networks can improve robustness to environmental changes, when trained on a single task. These results suggest that plasticity enables agents to adapt online, effectively providing a buffer against uncertainty. Building on this intuition, this thesis extends the use of plasticity from single-task settings to a distribution of environments. Specifically, we train a plastic agent across multiple tasks, allowing a meta-optimizer to learn task-appropriate Hebbian update rules. During deployment, these learned rules enable the agent to adapt its behaviour dynamically based on its interaction with the environment. The central question of this work is whether such meta-learned plasticity generalizes beyond the training distribution. In particular, we investigate whether the learned Hebbian update rules enable effective adaptation to previously unseen task variations. The goal of this work is to evaluate whether plasticity can serve as an effective mechanism for adaptation in environments that exhibit the characteristics of a big world.

1.4 Architectural Paradigms of Adaptation

While Hebbian plasticity provide a sound framework for online adaptation, implementing these mechanisms shifts the problem from algorithmic design to software engineering. Traditional machine learning architectures are built upon a strict decoupling of lifecycle phases: learning occurs offline during a resource-intensive training phase, resulting in a static artifact (a frozen model) that is subsequently deployed purely for inference. Under this classic MLOps paradigm, handling environmental changes or data drift requires an external adaptation loop, where production data is gathered, and used to entirely retrain and redeploy a new static model. As an engineering pipeline, this approach is highly automated but deeply inefficient, requiring the repeated recomputation of historical datasets to incorporate small amounts of new data.

Embedding self-modifying capabilities directly into the controller introduces a paradigm shift in software architecture. Instead of treating adaptation as a periodic, external lifecycle process, mechanisms like learned Neural Hebbian plasticity embed adaptation internally within the running system itself. This transition reduces the frequency of offline retraining pipelines, yet it introduces architectural trade-offs at runtime.

By allowing model parameters to drift dynamically during deployment, the system fundamentally departs from the predictable, deterministic behaviour expected in traditional software engineering. Identical inputs at different points in the system’s lifetime may result in different outputs, complicating standard validation, debugging, and reproducibility. Consequently, adopting internal

adaptation paradigms requires a restructuring of the MLOps lifecycle, shifting the engineering focus from frequent offline training to sophisticated runtime monitoring, and robust safety mechanisms.

1.5 Research Questions

This thesis investigates whether meta-learned Hebbian plasticity can improve generalization in an underparameterised approximation of the “Big World.” Rather than attempting to fully model the complexity of deployment, we construct a limited training regime through domain randomisation, exposing agents to a restricted but varied subset of possible environmental conditions. Within this setting, we examine whether learned online plasticity enables better performance.

To evaluate the effectiveness, scalability, and broader system engineering implications of this approach, we formalize our investigation into four core research questions. The first two focus on algorithmic adaptation and physical generalization, while the latter two investigate the software architecture and MLOps paradigms introduced by continuously adaptive systems.

Algorithmic Adaptation and Generalization

RQ1: Does meta-learned plastic networks outperform static networks in adapting to unseen environments

Context: We examine whether incorporating Hebbian plasticity allows a single network to dynamically adapt its control strategy in a locomotion task across diverse morphologies, ultimately outperforming traditional non-plastic baselines. In a setting where an agent is trained on a set of morphologies, we hypothesize that standard static neural networks will overfit to the exact body structures they were trained on, while plastic networks will be able to learn a robust locomotion strategy.

RQ2: Does increasing the expressivity of the local plasticity rule improve generalization?

Context: The Hebbian learning rule is often parametrized by only a handful of coefficients and without non-linearity. We investigate whether a more expressive learning rules with additional parameters and more functional flexibility can improve an agent’s ability to adapt. By comparing plasticity rules of varying complexity, we aim to determine whether increased expressivity leads to more effective online learning and improved task performance.

Architectural and MLOps Implications

RQ3: How does plasticity alter the MLOps lifecycle compared to traditional external retraining pipelines?

Context: Traditional MLOps architectures assume a static model post-deployment, relying on resource-intensive offline retraining loops to handle data drift. Moving adaptation directly into the runtime environment via plastic neural networks reduces retraining frequency but fundamentally reshapes the machine learning lifecycle.

RQ4: What architectural patterns and safety mechanisms are required to monitor, validate, and control model behaviour during continuous deployment?

Context: Because plastic networks continuously update their parameters in response to real-world interactions, identical inputs may no longer result in identical outputs, introducing severe challenges for predictability and debugging. We investigate the necessary software engineering safeguards to ensure the adaptation process works reliably.

2 Literature Review

2.1 Introduction

Hebbian learning mechanisms offer a biologically inspired alternative to conventional gradient-based meta-learning. In this survey, we review recent work on Hebbian plasticity in neural networks. We compare optimisation strategies, architectural choices, and application domains. By organising the field along key design dimensions, we aim to provide a conceptual overview and identify promising directions for future research on meta-learning and synaptic plasticity.

Hebbian learning is a mechanism by which the strength of the connection between two neurons is adjusted as a factor of their activity.

Biological systems excel at online continual learning in non-stationary environments and rapid adaptation to novel situations, conditions in which optimisation with stochastic gradient descent is suboptimal.

In biology, one of the primary mechanisms for learning is synaptic plasticity, first proposed by Hebb (1949). According to this model, synaptic updates occur based on local neuronal activity, often summarised by the phrase: "neurones that fire together wire together." see Brown et al. (1990) for a survey of synaptic plasticity in the brain.

This survey does not include work focused on biological brains, although many of the papers discussed here are inspired by, and explicitly reference, neuroscience research. Our focus is limited to work that aims to improve the capabilities of artificial intelligence systems. In addition, we exclude papers published before 2015; while somewhat arbitrary, this cutoff allows us to concentrate on recent developments in the field. Readers interested in a survey of earlier work are referred to Soltoggio et al. (2017) and Coleman and Blair (2012). We identify three broad clusters of work in the literature on Hebbian learning within the field of artificial intelligence. While these directions share a common foundation, they differ in motivation, methodology, and their relation to other research communities.

The first cluster explores biologically inspired alternatives to backpropagation for learning early representations. These approaches rely on local Hebbian plasticity rules to learn representations

in an unsupervised manner, while a final classification layer is trained using gradient-based methods. A representative example of this direction is Krotov and Hopfield (2018). The second cluster builds on the concept of fast weights, in which all synaptic connections are updated using a shared Hebbian rule. These updates function as a form of memory. See Ba et al. (2016) for an example of this approach. The final cluster focuses on Hebbian meta-learning and aims to create networks that can rapidly adapt to novel situations by leveraging learned update dynamics rather than relying solely on fixed weights. In this line of work, different plasticity rules are assigned to different connections in the network, and the parameters of these rules are themselves learned across tasks, resulting in highly expressive learning systems.

Our focus in this survey is on this last category, as it represents a promising direction for building systems capable of rapid adaptation and continual learning. This survey is intended as a general introduction to the topic.

2.2 Background

2.2.1 Learning to learn

The aim of meta-learning is to create systems that quickly adapt to novel tasks. In meta-learning two optimisation processes exist in a layered structure. An outer loop is responsible for tuning the parameters of an inner loop learning algorithm. The inner loop, is responsible for adapting to new tasks. The motivation is that efficiently fitting a model in high dimensional space requires strong inductive biases. Rather than handcrafting such biases, the meta learning approach seeks to acquire them automatically by capturing regularities across a task distribution. In other words, restricting the learning domain, rather than using general-purpose learning algorithms, improves efficiency.

In order to better understand what meta learning is, perhaps the simplest example of it is automatic hyperparameter search.

Several surveys on meta-learning such as Beck et al. (2024), Vettoruzzo et al. (2023), Huisman et al. (2021), and Tian et al. (2022) provide a good overview of the field. However, Hebbian meta-learning is either omitted or only briefly mentioned. This survey aims to address that gap by shedding light on this lesser known but promising line of research.

Various meta-learning approaches have been proposed, and they are typically categorized into three broad families: optimization-based, black-box (or model-based), and metric-based methods.

Metric-based meta-learning methods aim to learn a representation space in which examples

from the same class or task are close together, and those from different classes or tasks are far apart. Instead of updating model parameters during task adaptation, these methods rely on comparing new examples to a small set of labeled support examples using a learned similarity metric. A well-known example is Prototypical Networks Snell et al. (2017), which compute a prototype (i.e., class centroid) in embedding space for each class, and classify new inputs based on their distance to these prototypes.

A well-known optimization-based approach is Model-Agnostic Meta-Learning (MAML) Finn et al. (2017). In this method, the meta-learner optimizes the initial parameters of a neural network such that it can quickly adapt to new tasks using just a few gradient descent steps. The goal is not to learn a specific update rule, but rather to find an initialization that generalizes well across a distribution of tasks.

In contrast, black-box methods learn to adapt internally by using a meta-learner such as a recurrent network, hypernetwork, or plastic network (e.g., Hebbian). These models do not perform explicit inner-loop gradient updates. Instead, they adapt by updating their internal state or generating new weights during inference based on the data they observe. The learning process is encoded implicitly in the architecture or dynamics of the network.

2.2.2 Hebbian Learning and Synaptic Plasticity

In this section we will review the basics of Hebbian learning. In its simplest form Hebbian learning strengthens the connection between two neurons if both are active.

Vanilla Hebbian learning rule

$$w_{i,j}(t+1) = w_{i,j}(t) + \Delta w_{i,j} \quad (2.1)$$

$$\Delta w_{i,j} = \eta a_i a_j \quad (2.2)$$

Where η is the learning rate, a_i and a_j denote the activations of the *pre-synaptic* and *post-synaptic* neurons, respectively, that is, the neurons on either side of the connection. This rule is typically applied at every time step to every weight in the network. This learning rule is simple and captures the core idea of Hebbian learning: neurons that fire together, wire together. However, it is rarely used as is, in practice because of the problem of unbounded weight growth (or decay). The instability arises from a feedback loop in which correlated activity strengthens the weights, which in turn increases future correlations. There are various methods to address this, such as adding a decay term.

Vanilla + decay term To implement this, we introduce a hyperparameter λ (between 0 and 1) into equation (1):

$$w_{i,j}(t+1) = \lambda w_{i,j}(t) + \Delta w_{i,j} \quad (2.3)$$

This approach has the disadvantage that, if a set of neurones is not frequently activated, the weights and therefore the memories they encode will gradually fade to zero.

Weight clipping In this simple method we simply clamp the weights to a range for example $[-1,1]$

$$w_{i,j}(t+1) = CLAMP(w_{i,j}(t) + \Delta w_{i,j}) \quad (2.4)$$

Synaptic Competition In this method we apply an additional step after each weight update

$$w'_{i,j} = \frac{w_{i,j}}{\|\mathbf{w}_i\|_2}$$

By normalizing the vector of incoming synaptic weights, all connections to a neuron must share a fixed-length budget. As some synaptic weights grow, others are necessarily reduced to maintain the unit norm.

Besides the vanilla learning rule, different variants of Hebbian learning exist, we will review some commonly used ones here.

Oja's rule Originally introduced by Oja (1982), Oja's rule maintains stable weight values even in the absence of stimulation, thereby enabling the storage of long-term memories while preventing unbounded weight growth. It can be expressed compactly as:

$$\Delta w_{i,j} = \eta a_j (a_i - a_j w_{i,j}(t)).$$

Expanding this expression shows explicitly that Oja's rule consists of a Hebbian term and a stabilizing decay term. Because the decay term is proportional to the post-synaptic activity a_j . If the neuron does not fire ($a_j = 0$), the weight update $\Delta w_{i,j}$ vanishes entirely, leaving the weight unchanged.

$$\Delta w_{i,j} = \eta a_j a_i - \eta a_j^2 w_{i,j}(t).$$

ABCD model

$$\Delta w_{ij} = \eta \cdot (Aa_i a_j + Ba_i + Ca_j + D)$$

This is a commonly used generalization of the vanilla Hebbian update. Each term can be interpreted as follows:

- A modulates the standard Hebbian term, scaling the contribution of the correlation between the pre- and post-synaptic activations.
- B scales the contribution of the pre-synaptic activation alone, independent of the post-synaptic neuron.
- C scales the contribution of the post-synaptic activation alone.
- D is a constant bias term, which can be interpreted as an excitatory or inhibitory baseline adjustment to the weight.

This method not only provides a stable learning rule but can also express more complex relationships between pre- and post-synaptic activity.

In essence the choice of how to parameterise the learning rule is determined by the trade-off between computational resources and expressive capacity, and the parallel bias-variance trade off.

2.3 Plasticity in Meta-Learning: A Conceptual Overview

Now we have seen how Hebbian plasticity works in detail but what does it do at a conceptual level and how is it useful? The update rule detects and reinforces patterns of co-activation between neurons. When a group of neurons is strongly co-active during learning, their mutual connections are strengthened, making the network more sensitive to familiar patterns. In future situations, even partial or noisy activation of that pattern will elicit a strong response. In essence, Hebbian plasticity allows the network to store and recall correlations between inputs. The connection weights become a memory of which neurons tend to be active together. Note, however, that the vanilla Hebbian learning rule is very simple, and more sophisticated models (such as the ABCD model) allow for learning rules with a negative sign (anti-Hebbian). This promotes decorrelation of activation patterns

Despite the diversity of synaptic update rules, the shared essence of Hebbian learning is characterised by 3 key principles: first, it depends only on local information, which means the activity of the two neurones connected by the synapse, rather than the global state of the network. Second, it

is unsupervised, with no input from a loss signal or reward feedback. Lastly, it is online, the weight updates occur at each new input, rather than needing a batch of data.

In Hebbian meta-learning each synaptic connection in the network is assigned its own learning rule. This greatly increases the expressivity of the learning algorithm that can be implemented, compared to non meta-learning approaches such as fast weights, where the parameters of the update rule are shared globally. In the meta-learning setting, one or multiple terms of a parametric weight update, such as the learning rate η or the ABCD terms, is learned across tasks. The set of all connection specific weight update rules effectively define a learning algorithm that can optimise a network to perform well even if it receives no feedback or reward at test time. That is because the feedback is taken into account at the meta level in the outer loop.

It has been argued that such networks implicitly learn to discover reward-predictive cues during meta-training Ben-Iwhiwhu et al. (2020) Because of this, it is related to methods that aim to meta-learn a loss function such as Bechtle et al. (2021). Nevertheless, incorporating the reward or loss signal directly into the input, an approach used in other meta-learning methods like Duan et al. (2016) and Wang et al. (2016), is likely to improve performance, as it enables the model to condition its behaviour more directly on task feedback.

2.4 Advantages and Limitations

The strong point of Hebbian meta-learning is that it increases the robustness of the policy. It has shown to perform well when generalising to unseen conditions, work by Najarro and Risi (2020), demonstrated this by training agents on a quadrupedal locomotion task; while static networks failed to adapt to unseen structural changes, agents utilizing evolved Hebbian learning rules adapted to novel morphologies, such as a damaged limb not encountered during training, without requiring an explicit reward signal. Building on this, Leung et al. (2025) show that Hebbian-based plastic networks can have zero-shot sim-to-real transfer capabilities and can adapt to unseen situations such as irregular terrain and novel morphologies. Finally, Ferigo et al. (2023) further demonstrated that evolving Hebbian learning rules in Voxel-based Soft Robots enables lifetime learning, where agent performance consistently improves as synaptic weights self-organize in response to environmental stimuli. Their results indicate that plastic controllers are significantly more resilient to unforeseen morphological damage and generalize more effectively to unseen hilly terrains than static baselines have demonstrated that when trained on a single task, networks with plasticity adapt better to unseen environmental changes than static networks. This suggests that plasticity provides an inherent "buffer" against uncertainty.

even when parts of the policy network were zeroed out Najarro and Risi (2020).

Architecture. Across the surveyed works, most models rely on relatively simple feedforward or recurrent neural networks, such as MLPs, RNNs, and LSTMs. Overall synaptic plasticity can be implemented with any network topology.

Weight Initialization Some papers apply Hebbian learning to a pre-trained network, as in Miconi et al. (2018). In this approach, both the fixed base weights and the parameters of the Hebbian update rule are co-optimized during meta-training. An advantage of this approach is that the network weights require less task-specific adaptation, as they are already initialized in a useful configuration. This setup is similar to the MAML method Finn et al. (2017) and can be compared to inducing a form of "innate behaviour".

Another alternative is to apply Hebbian learning to networks that are initialised with different random weights at each episode, as in Najarro and Risi (2020), or to networks with all weights set to zero, as in Ferigo et al. (2023).

In these approaches, only the weight update rule is learned during meta-training. Starting from a new random network for each task offers increased robustness, as the learning rule cannot simply memorize a fixed trajectory in weight space. Additionally, from a research perspective, this setup helps disentangle the effects of weight initialization from those of the learning rule itself. In the two papers cited above, plastic networks were compared to standard networks on reinforcement learning tasks and achieved comparable performance at test time, despite having to effectively train a network from scratch during each episode. This result demonstrates the notable sample efficiency that Hebbian learning mechanisms can achieve.

Meta-Optimiser The optimisation algorithm is not used to optimize the weights directly but instead it is used to find the set of Hebbian coefficients that will control the weights of the network during its runtime based on the inputs.

The parameters of Hebbian learning rules are often meta-learned with evolutionary strategies (ES) because these black-box optimization methods have the advantage of not requiring the back-propagation of gradients, which can be quite complicated due to the indirect effects of meta-parameters. The particular ES often used is Salimans et al. (2017), a population-based optimization algorithm which has shown to reach competitive performance compared to other deep reinforcement learning approaches across a variety of different tasks. It also scales very well to large populations and large numbers of parameters.

A significant advancement was the introduction of differentiable plasticity, which enabled the use of gradient descent to optimize plasticity parameters. The theoretical feasibility of this approach was first demonstrated in Miconi (2016) and shown to scale to large (millions of parameters) networks in Miconi et al. (2018). Gradient based methods allows us to leverage the power of modern deep learning. The relative trade-offs and advantages between ES and gradient based methods is not quite clear, and it could be useful to compare them with respect to sample efficiency and other metrics. The comparative trade-offs between Evolutionary Strategies (ES) and gradient-based methods for this specific application are not yet fully established, an empirical comparison focusing on sample efficiency and other performance metrics could be an informative work for researchers in this domain.

The Information Bottleneck Hypothesis The genomic (or information) bottleneck hypothesis is inspired by the observation that, in biology, the information stored in the genome is several orders of magnitude smaller than what is needed to describe the wiring of the brain. In this analogy, the genome contains the meta-parameters optimised by natural evolution while the wiring of the brain are the base parameters optimised by an inner loop algorithm. Meta-learning Hebbian plasticity rules typically requires many more meta-parameters than base parameters, for example 5 meta-parameters for each neural connection (i.e the $\eta ABCD$ terms). This lets each connection in the network implement a highly specific learning rule but there are downsides. Firstly it is computationally expensive; optimising meta parameters is inherently costly, as each meta-parameter update requires running many updates of the base parameters. With a 5 to 1 meta-to-base parameter ratio, it potentially becomes prohibitive to tackle problems more complex than simple toy tasks. The second more fundamental issue, is that an overparameterised learning algorithm likely does not generalise well, that is, the inner loop Hebbian rules don't implement a general learning algorithm but overfit and implement a highly specific learning rule that can't adapt the weights to tasks not seen during meta-training. The hypothesis is that limiting the number of meta-parameters through a "genomic bottleneck" acts as an implicit regulariser, leading to better generalisation. Palm et al. (2020) test the hypothesis by sampling each synapse's learning rule from a Gaussian Mixture Model (GMM), while meta-learning both the parameters of the GMM and the cluster assignment distributions for each synapse, but without successfully validating the hypothesis. However Pedersen and Risi (2021) are able to reduce the of number learning rules and improve generality by periodically clustering similar learning rules and merging them into a shared rule during training. Wang et al. (2022) and Ferigo et al. (2024) propose a simple yet effective method where instead of assigning a unique rule per connection they assign plasticity parameters per neuron, greatly improving the

scalability.

Activity Traces Another source of local information that we can leverage, besides current activations, is the history of previous activations. Ferigo et al. (2024) propose a "weightless Hebbian Learning" method where they store a record of previous activations for each neurone, and use it to approximate the weights of each connection without having to store them explicitly, this greatly reduces memory usage. Yaman et al. (2019b), Yaman et al. (2020a) and Bizzotto et al. (2021) use the frequency of co-activations accumulated over an episode to perform synaptic updates, incorporating a form of local memory into the learning process. However, the method is coarse-grained with weight updates occurring only at the end of each episode. Rodriguez et al. (2022) incorporates the value of the plastic component from the previous time step into the update rule, scaled by a synapse-specific, learned decay rate. By enabling synapses to not only learn at different rates but also forget at different rates, this approach allows the network to adaptively balance stability and plasticity, improving its ability to learn temporal dependencies and retain relevant information over varying timescales. Darlow et al. (2025) use the correlation of activation histories between neurones over time as a latent representation which is then used as part of a broader mechanism for generating outputs. While Hebbian in spirit, this does not constitute a true plasticity mechanism because synaptic weights are not modified.

Neuromodulated Plasticity Neuromodulation refers to the mechanism by which a neural system dynamically controls when and where plasticity occurs, it helps selecting important information while filtering irrelevant information, thereby mitigating catastrophic forgetting of previously learned knowledge. Neuromodulation works by replacing the fixed Hebbian learning rate with a signal computed by specialised *neuromodulator* neurones or set of neurones in the network, allowing the model to control how much learning occurs at each time step. This self-modifying capability enables networks to adapt more efficiently in dynamic environments and to retain information.

To implement neuromodulation we replace the learning rate η in the update rule with a modulatory signal $M(t)$ (we take the vanilla update rule for simplicity, of course it can be applied to any variant)

$$\Delta w_{i,j} = M(t) a_i a_j \quad (2.5)$$

The modulatory signal can be global as in Duan et al. (2023), synapse or neuron specific as in

Miconi et al. (2019) or specified layer-wise as in Wang et al. (2022).

Unlike the classical Hebbian postulate where synaptic strength is governed solely by the correlation between presynaptic activity x_i and postsynaptic activity y_j , this rule introduces a third factor M . This signal, typically representing a reward or a reward-prediction error such as dopamine, acts as a gating mechanism that determines whether the local neural coincidence should be encoded into long-term structural changes. When M is high, the system "validates" the recent pairing of neurons as being behaviourally relevant, thereby strengthening the connection; conversely, if M is absent or negative, no lasting plasticity occurs. This method effectively bridges the gap between local synaptic events and distal global outcomes, allowing the network to selectively retain only those internal associations that lead to successful environmental interactions.

2.5 Applications

In this section, we briefly review the key application domains where meta-learning with Hebbian plasticity has been explored. These domains are particularly interesting because they leverage the ability of plasticity rules to adapt rapidly and efficiently, mimicking aspects of biological learning.

2.5.1 Continual Learning

Continual learning requires a model to acquire new tasks sequentially without catastrophic forgetting of previously learned tasks. Hebbian plasticity can help by reinforcing frequently co-activated pathways while allowing uncorrelated or less-relevant connections to decay. Meta-learning approaches in this domain adapt the parameters of the Hebbian update rules themselves (such as learning rates or decay rates), allowing the network to dynamically adjust its plasticity based on task demands. This flexibility enables the integration of new information while preserving existing knowledge, a key requirement for continual learning.

2.5.2 Few-shot Learning

Few-shot learning tasks require a model to generalize from only a few examples of a new class or task. Hebbian plasticity offers a natural way to rapidly encode correlations from limited samples, enabling the network to quickly form new representations. In meta-learning settings, the parameters of the plasticity rules are themselves learned, so the network can meta-learn how to best incorporate sparse, new examples into its memory. This mirrors biological mechanisms where synaptic changes are fast and local, facilitating rapid adaptation.

2.5.3 Reinforcement Learning

RL is well suited for Hebbian learning because RL is sequential; in the classical supervised learning setting, independent and identically distributed samples are assumed. In contrast, reinforcement learning (RL) involves highly correlated samples collected sequentially through interaction with the environment. This temporal correlation presents challenges for learning and stability. Hebbian plasticity can help mitigate this by locally tracking correlations over time (e.g., eligibility traces), providing a more stable way to assign credit to recent activity. Meta-learning can further enhance this by adapting the plasticity rules themselves, tuning how strongly recent activations contribute to weight updates or how quickly traces decay. This flexibility can improve performance in RL tasks by enabling the agent to better track temporally correlated information and adapt to changing environments.

2.6 Survey of Existing Work

Table 2.1: Comparison of papers on plasticity in meta-learning.

Ref	Domain	Meta Optimiser	Model type	Learning rule	Neuromodulation
Miconi (2016)	One-shot learning	Gradient descent	MLP	Vanilla	No
Orchard and Wang (2016)	RL	Evolution	RNN	Other	No
Miconi et al. (2018)	RL, computer vision	Gradient descent	RNN	Oja	No
Yaman et al. (2019a)	RL	Evolution	RNN	Other	Yes
Wang and Orchard (2019)	RL	Evolution	RNN	Other	No
Najjarro and Risi (2020)	RL	Evolution	MLP	ABCD	No
Miconi et al. (2020)	RL, NLP	Gradient descent	LSTM	Other	Yes
Thangarasa et al. (2020)	Continual learning, computer vision	Gradient descent	MLP, CNN	Vanilla	No
Ben-lwhiwhu et al. (2020)	RL	Evolution	RNN	ABCD	Yes
Palm et al. (2020)	RL	Evolution	MLP	ABCD	No
Yaman et al. (2020b)	RL	Evolution	RNN	Other	No
Pedersen and Risi (2021)	RL	Evolution	MLP	ABCD	No
Yaman et al. (2021)	RL	Evolution	MLP	Other	Yes
Bizzotto et al. (2021)	RL	Evolution	RNN	Other	No
Rodriguez et al. (2022)	RL	Gradient descent	RNN	Other	No
Wang et al. (2022)	RL	Evolution	RNN	ABCD	Yes
Duan et al. (2023)	Few-shot learning, computer vision	Gradient descent	RNN	Vanilla	Yes
Ferigo and Iacca (2023)	RL	Evolution	MLP	ABCD	No
Ferigo et al. (2023)	RL	Evolution	MLP	ABCD	No
Miconi (2023)	RL	Evolution	RNN	Other	Yes
Ferigo et al. (2024)	RL	Evolution	MLP	ABCD	No
Leung et al. (2025)	RL, sim-to-real	Evolution	MLP	ABCD	No
van Diggelen et al. (2025)	RL	Evolution	MLP	ABCD	No

2.7 The Case for Hebbian Plasticity: A Comparison with Other Meta-Learning Approaches

Within any given task distribution, there are, on the one hand, invariant features, elements that remain constant across tasks—and, on the other hand, task-specific features that vary between tasks. For an agent to perform well, it must internalize the invariant structure as prior knowledge, while still being able to rapidly adapt to task-specific variations.

Miconi et al. (2018) frame this distinction in terms of storage and computation. A neuron whose weights remain fixed during a task can be interpreted as performing a storage or memorization function. In contrast, a plastic neuron, whose weights change during task execution, can be seen as performing computation. This perspective can also be described as a distinction between constant and variable neurons.

This trade-off is closely related to the stability–plasticity dilemma in continual learning: emphasizing stability limits the network’s ability to acquire new knowledge, while excessive plasticity can lead to catastrophic forgetting.

A key advantage of trainable synaptic plasticity is its flexibility. Unlike other approaches, it does not require specifying in advance a fixed ratio between stable and plastic elements. Instead, each parameter can have its own learned learning rate, allowing the inner-loop learning dynamics to adapt in a highly specific way to the task distribution. For example in *RL*² the plastic parameters correspond to the size of the (usually quite small) hidden state, at the other extreme, methods such as hypernetworks map each task to an entirely new set of parameters. Both approaches impose strong assumptions about how much adaptation should occur, in both cases the degree of plasticity is effectively fixed by design. Manually selecting an appropriate stability–plasticity trade-off at the architectural level is therefore likely to be brittle, as the optimal balance depends on the task distribution. In contrast, meta-learning plasticity at the level of individual parameters provides a more scalable and adaptive solution. For example, if tasks are highly similar, most parameters may converge to very small (or zero) learning rates, effectively becoming stable. Conversely, for more diverse task distributions, higher learning rates allow for greater adaptability. In essence, this means that the inner-loop learning algorithm can be shaped and specialized through meta-learning.

3 Proposed Method

This chapter describes the control methods evaluated in this thesis, the underlying system architecture used to train them, and the design trade-offs associated with different forms of within-episode adaptation. The next chapter then describes the experimental setup used to evaluate these methods.

3.1 Universal Control Across Embodiments

The challenge of training a universal control policy, with the ability of zero shot generalization to unseen robot morphologies is a theme in recent robot learning literature, notably explored in frameworks like MetaMorph Gupta et al. (2022) and MetaLoco Zargarbashi et al. (2024). While this thesis focuses on limb length variants within the MuJoCo Ant environment, MetaMorph takes this a step further by using Transformer-based architectures to treat different robot morphologies as sequences of parts, allowing a single policy to control vastly different skeletal structures. Similarly, MetaLoco addresses the generalization gap by combining Meta-Reinforcement Learning (Meta-RL) with motion imitation. Unlike standard RL approaches that might struggle with the physics of unseen limbs, MetaLoco leverages prior "expert" motion data to provide a stylistic baseline, then uses meta-learning to rapidly adjust torque outputs based on the specific dynamics of the quadruped at hand. By situating this experiment alongside these works, the goal shifts from simply "moving forward" to developing a universal controller capable of interpreting its own physical constraints as a variable in the decision-making process.

3.2 Models Compared

Four different models were evaluated in this study, they are described below.

Feedforward Network A fully connected feedforward neural network with two hidden layers of 64 and 32 units, respectively, tanh (hyperbolic tangent) activation functions. This standard network serves as a baseline.

RL^2 Reinforcement Learning Squared (RL^2) is a meta-reinforcement learning method that implements the idea of “learning to learn” by representing the agent’s policy as a recurrent neural network. In RL^2 , the agent’s input is augmented with information from the previous time step, including the previous action and reward. This allows the recurrent network to use its internal hidden state to develop task-specific adaptation strategies. In this work, I use an adapted version of RL^2 for single-episode learning. The original method is designed to support adaptation across multiple episodes, where the agent can use experience from earlier episodes to improve its behaviour in later ones.

To ensure a correct implementation, I reproduced the original RL^2 paper in Appendix A.

Hebbian Plastic Network with ABCD Rule A fully connected neural network with two hidden layers of 64 and 32 units, respectively, using tanh activation functions and with no bias terms. In contrast to the baseline, weights are dynamically updated during the episode via a learned Hebbian plasticity rule.

Each layer maintains a set of fast weights that evolve according to a parameterized Hebbian update following the ABCD rule (see survey chapter) combining pre- and post-synaptic activity, scaled by a learnable learning rate.

Hebbian Plastic Network with Neural Update Rule It is also possible to use neural networks to parameterize the update rule. In this approach, a separate neural network is trained to generate the weight update $\Delta w_{i,j}$.

$$\Delta w_{i,j} = f_{\theta}(a_j, a_i),$$

where f_{θ} is a neural network with learnable parameters θ .

This meta-learning approach allows the network to learn complex, task-specific update rules that can generalize beyond hand-designed plasticity mechanisms.

A fully connected neural network with two hidden layers of 16 and 8 units, respectively, using tanh activation functions and with no bias terms. As in the Hebbian plastic network, weights are dynamically updated during the forward pass via a learned plasticity rule.

In contrast to the ABCD-based approach, each synapse is equipped with an independent small multilayer perceptron (MLP) that computes weight updates based on local pre- and post-synaptic activity. This approach is introduced here as an exploratory extension to investigate whether more expressive, non-linear parametrizations of plasticity rules provide additional benefits over standard

formulations.

Concretely, each synapse-specific MLP takes the pre- and post-synaptic activities as input and consists of two hidden layers with 3 units each and tanh activations.

3.3 Comparison of Model Complexity and Capacity

Table 3.1: Comparison of outer-loop optimized and inner-loop adaptive parameter count across models.

Component	Baseline FF	RL^2	Hebbian (ABCD)	Neural Hebbian
Outer-loop optimized parameters	4136	25896	20160	15600
Inner-loop adaptation (within episode)	None	64	4032	624

The comparison in Table 3.1 highlights a key architectural trade-off between meta-learned capacity and within-episode adaptability. The distribution of these parameters varies across the evaluated methods, reflecting different structural assumptions about how an agent should balance long-term meta-knowledge against rapid, online adaptation. The Hebbian ABCD agent exhibits a balanced 5:1 ratio of outer-loop optimized parameters to inner-loop adaptive parameters. In contrast, the Neural Hebbian model shifts heavily toward outer-loop allocation with a 25:1 ratio. While parameterizing the update rule via a small MLP dramatically enhances its expressive power, it massively inflates the outer-loop overhead. This heavy footprint, making it prohibitive to scale, as a result practical deployment necessitates restricting the Neural Hebbian approach to a smaller base network. An important difference is that the ABCD rule is inherently rigid, its ratio mostly fixed by its algebraic formulation, though minor modifications are possible, such as dropping the bias (D) term to trim outer-loop complexity. The Neural Hebbian model, however, offers an expansive range of architectural freedom. By altering the width, depth of the parameterizing MLP, researchers can dynamically tune this ratio across a wide spectrum to fit specific task demands. For the RL^2 method, the inner-loop parameters consist solely of the hidden state of the recurrent network. The feed forward baseline, with its fixed weights serves as a touchstone for determining whether adaptation within the episode is necessary at all for the given task. To ensure a fair comparison, the network sizes were chosen to approximately match the total number of trainable parameters across all methods. Given these vastly different parameter allocations, a systematic investigation exploring whether a ‘sweet spot’ ratio exists across different environments and methods is a promising direction for future research.

3.4 Architectural Complexity and Software Engineering Trade-offs

The parameter distributions detailed in Table 3.1 highlight a fundamental challenge in hardware resource allocation. In modern machine learning, deploying a model within the constraints of GPU or TPU memory (VRAM) is a zero-sum game. When designing an architecture, engineers must decide how to allocate this finite memory budget between static, compile-time knowledge and dynamic, run-time adaptability. The Neural Hebbian model represents an extreme allocation strategy, dedicating a massive portion of its capacity to the outer-loop update mechanism rather than the agent's primary policy. Because the MLP parameterizing the learning rule consumes so much memory, the system starves the base network of resources. This structural bottleneck is exactly why practical deployment of the Neural Hebbian approach forces the use of a smaller base network; the hardware resources are eaten up by the infrastructure of learning rather than the capacity to act.

This allocation of resources also has a severe impact on the system's operational latency, effectively tanking the model's framerate during deployment. In a standard feed-forward model or a recurrent baseline like RL^2 , a forward pass is a highly optimized, single-shot mathematical operation. However, introducing inner-loop plasticity fundamentally changes the execution cycle. The system must first perform standard inference, calculate the localized updates for every synapse, and then physically overwrite those weight matrices in memory at every single time step. This dual-phase execution cycle drastically increases the latency per step. From a systems engineering perspective, this makes highly plastic models difficult to deploy in real-time environments, such as high-frequency robotics, where the system operates on a strict latency budget and must maintain a high, deterministic frame rate to function safely.

Finally, this inner-loop Hebbian plasticity creates a hurdle for system scalability by breaking standard batch inference pipelines. In typical reinforcement learning setups, engineers scale up training and deployment by running dozens or hundreds of environments in parallel. This is efficient because observations from all environments can be batched together and passed through a single, shared set of static network weights running on a single chip. However, when the weights themselves adapt continuously within an episode, the entire network becomes a unique, localized state. A system can no longer route a batch of parallel environments through one shared matrix; instead, it must instantiate and maintain a separate, independent weight matrix for every single environment in the batch. This causes the memory footprint to explode linearly with the number of parallel workers, turning a highly parallelizable architecture into a severe memory and bandwidth bottleneck.

This is an instance of the "hardware lottery" Hooker (2021). According to the author contemporary deep learning architectures have succeeded in large part because they are perfectly optimized

for hardware accelerators designed to execute massive, uniform matrix multiplications on static weights. The industry in turn increasingly pivots toward domain-specialized hardware designed to accelerate these specific algorithm. As a result of this hardware-software co-evolution there is a risk of painting ourselves into a corner.

By tying within-episode adaptation directly to weight mutations rather than transient activation states, as is standard practice in Recurrent Neural Networks and Transformers, Hebbian architectures run counter to the current hardware paradigm. Consequently, these models face an architectural penalty. This is not an inherent algorithmic flaw, but rather a mismatch with traditional von Neumann architectures, where memory and computation are separated. Realizing the full efficiency of Hebbian plasticity requires processing-in-memory hardware paradigms. Unlike standard GPUs, which suffer severe bandwidth penalties when constantly moving dynamic weight data between separate memory and processing units, these alternative architectures physically co-locate compute and memory directly at the "synapse" level. This physical colocation natively supports continuous, localized weight updates, allowing highly plastic models to scale efficiently without relying on the rigid, uniform batching pipelines required by modern accelerators.

3.5 System Architecture for Massive Parallelization

3.5.1 Computational Challenges of Meta-Learning

Meta learning is inherently very computationally expensive because it requires indirectly optimising a learning algorithm over a distribution of tasks. In the case of Evolution Strategies (ES), each generation requires evaluating thousands of candidate parameter sets across multiple environments. With a population size of $P=4096$, sequential execution becomes impractical due to excessive simulation and policy evaluation costs. With a population size of $P=4096$, sequential execution becomes impractical and lengthy.

3.5.2 Unified GPU-Based Execution Pipeline

To manage this computational constraint, the entire pipeline was implemented using the JAX framework Bradbury et al. (2018). This architecture allows for end-to-end vectorization (via `jax.vmap`) and Just-In-Time (JIT) compilation of the simulation and the entire training loop. Vectorization is the process of expressing computations as batched array operations instead of explicit Python loops. The `jax.vmap` transformation allows many evaluations to be executed simultaneously rather than sequentially, utilizing the large number of processing cores available on GPUs.

Just-In-Time compilation is used through `jax.jit`. This compiles pure functional style functions (no side effects, no global variables), into optimized XLA operations. As a result, repeated Python-level interpretation is avoided, and the rollout, policy evaluation, fitness computation, and strategy update can be executed as highly optimized GPU operations.

As an illustrative example, the official JAX documentation shows that, once compilation time is amortized, a JIT-compiled function can execute approximately 30 times faster than an equivalent NumPy Jax Docs (2024). Therefore, for functions that are called frequently and can be written in a functional style, the additional compilation overhead is usually well worth the performance improvements.

Together, `vmap` and `jit` allow the full ES training loop to be represented as a single parallel computation graph. This is essential for scaling to large population sizes, because the computational cost is shifted from sequential CPU-controlled execution to batched accelerator execution.

A key feature of this implementation is that it bypasses the CPU-GPU communication bottleneck typically found in traditional Reinforcement Learning setups.

3.5.3 Accelerator(GPU) Native Components

To achieve efficient large-scale parallelization, all major components of the training pipeline were implemented using libraries designed to operate natively within the JAX ecosystem. Rather using the usual setup where simulation is executed on the CPU, and policy inference on the GPU. Here the entire computation is compiled into a unified execution graph and executed directly on the accelerator. This architecture minimizes communication overhead and allows all stages of training to remain within the same XLA device memory space. The architecture is composed of the following components:

- **MJX (MuJoCo JAX):** The physics simulation is fully offloaded to the GPU, enabling the simultaneous rollout of the entire population across multiple distinct environments Todorov et al. (2012).
- **Flax:** The neural network is implemented using the Flax Heek et al. (2024) functional parameterization. Unlike object-oriented deep learning frameworks, Flax treats the neural network as a pure function: $f(\theta, s) \rightarrow a$. By mapping this function over the population axis using `jax.vmap`, we achieve massive parallelization where 1,024 unique policies are evaluated in a single hardware call, ensuring that the evolutionary optimisation remains computationally tractable.

- **Evosax:** The evolutionary optimization process itself is executed on the hardware accelerator. By using evosax Lange (2022), the generation of new parameter candidates and the updating of the strategy’s state occur directly in GPU memory.

Together, these components form a fully accelerator-resident training pipeline in which simulation, policy evaluation, and evolutionary optimization are executed as a single computational workflow. This design enables efficient scaling to large population sizes and makes meta-learning computationally feasible for complex environments.

The complete source code used in this thesis is available at:

<https://git01lab.cs.univie.ac.at/marlony23/master-thesis/>

4 Experimental Setup

4.1 Description of experiment

This experiment is based on the MuJoCo Todorov et al. (2012) Ant environment, a widely used benchmark in reinforcement learning. The environment consists of a four-legged agent (the “ant”) controlled via continuous torque inputs applied at its joints. The agent operates in a simulated physics setting and is tasked with learning locomotion behaviours, aiming to move forward as efficiently as possible. Starting from the original Ant morphology, a set of 15 variants was generated by modifying the lengths of the limbs. These variants introduce structural diversity while preserving the overall topology of the agent. The resulting morphologies were divided into a training set consisting of 4 morphologies and a test set comprising 11 unseen morphologies. This split enables the evaluation of the agent’s ability to generalize beyond the specific body configurations encountered during training. This setup prevents the agent from relying solely on memorized gait patterns tied to a fixed morphology. Instead, successful performance requires learning a control policy that adapts to variations in body structure, thereby demonstrating true generalization across morphologies.

4.2 Choice of environment

The selection of a suitable environment was driven by two primary requirements: the need for task diversity to test Hebbian plasticity, and the technical requirement for massive parallelization to ensure computational feasibility. Recent literature suggests that networks equipped with synaptic plasticity rather than fixed weights, exhibit superior adaptation to unseen morphologies and environmental shifts. Following this intuition, this thesis shifts the focus from mastering a single static task to performing across a distribution of environments. By training on a sampled set and evaluating on an unseen set from the same distribution, we can quantify the “adaptive dividend” provided by the plasticity rules. To evolve or train these adaptive rules effectively, the environment must either have a set of variants available or support procedural generation of variants, that is the ability to create infinitely varied tasks to prevent over-fitting. The second requirement is vectorized

throughput: To iterate on plasticity rules, the simulation must run at a massive scale, ideally 10^6 or more steps per second, leveraging GPU acceleration to simulate thousands of environments in parallel. Several environments were considered, and are described along several dimensions in the table below.

Kinetix Kinetix Matthews et al. (2025) provides a diverse set of 2D physics-based environments that combine control, reasoning, and puzzle-solving. Its procedurally generated virtually endless different tasks make it useful for evaluating agents that must generalize beyond fixed benchmark settings. Because it is designed for efficient parallel simulation, Kinetix is particularly suitable. However, it also represents a higher level of difficulty, because the task distribution is very diverse, which is why it was not selected for this thesis.

MuJoCo XLA MuJoCo XLA Todorov et al. (2012), often referred to through MJX, extends the widely used MuJoCo physics engine with support for accelerated, array-based simulation. This makes it especially relevant for continuous-control and locomotion tasks, where agents must learn stable and coordinated movement in physically realistic environments. Its main advantage is that it combines the established MuJoCo benchmark tradition with more scalable training through hardware acceleration. MJX does not provide procedural environment generation out of the box. Therefore, in this work, an existing MuJoCo environment was modified to enable the generation of multiple environment variants, as described in more detail in subsection 4.3.

XLand-MiniGrid XLand-MiniGrid Nikulin et al. (2023) is a procedurally generated grid-world benchmark designed for studying generalization and meta-reinforcement learning. The environments are simpler than full physics simulators, but they provide a large variety of tasks, rules, and goals. This makes XLand-MiniGrid useful for testing whether agents can infer task structure and adapt their behaviour across changing problem settings. XLand-MiniGrid is more suitable for evaluating reasoning abilities and hard exploration challenges. In contrast, plastic networks have so far been shown to perform particularly well on continuous control tasks, such as robotic control. For this reason, XLand-MiniGrid was not selected as the environment for this experiment.

Unimal The Unimal dataset Gupta et al. (2021) introduces a universal and expressive representation for generating diverse robot morphologies. Instead of restricting variation to simple parameter changes, UNIMAL defines agents as kinematic trees composed of a central body and articulated limbs, allowing both structural and parametric modifications. This includes operations such as

adding or removing limbs, as well as changing their physical properties and joint configurations. As a result, the generated morphologies exhibit significant diversity in body layout, degrees of freedom, and locomotion strategies. While this enables the study of generalization across fundamentally different embodiments, it also significantly increases difficulty, as the underlying locomotion mechanisms may differ completely across morphologies. As a result, it is likely necessary to incorporate feedback signals, such as reward into the update rule to solve this challenge, or include the robot morphology as part of the observation as in Gupta et al. (2022). While Unimal is not natively vectorized, it could be ported to MJX, given its foundation on MuJoCo.

Table 4.1: Comparison of simulation environments for evaluating plastic neural networks and universal controllers.

Environment	Core Focus	Vectorized	Proc. Gen.
Kinetix	2D physics-based puzzles	Yes	Yes
MuJoCo XLA	3D locomotion	Yes	No
X-Land	Grid-world puzzles	Yes	Yes
Unimal	3D locomotion	No	Yes

Overall, the MuJoCo XLA environment was chosen because it offers high-speed vectorized throughput and continuous control complexity. While other environments provide greater task diversity or procedural generation out of the box, they either lack the simulation speed required for large-scale evolution (Unimal) or focus on discrete reasoning tasks less suited for demonstrating Hebbian adaptation (XLand-MiniGrid). While Kinetix could be a strong second choice due to its procedural diversity, its higher difficulty and task variety fell outside the primary scope of this study.

4.3 Morphology Generation

We generated variants of the MuJoCo Ant by modifying the XML model (ant.xml). Each morphology was defined by 8 discrete parameters controlling the lengths of the upper and lower segments of the four legs (short, medium, long). For each configuration, the XML file was programmatically updated by rescaling the corresponding leg segments while preserving the original body structure and joint alignment. We generated a small set of variants resulting in 15 distinct ant morphologies. Each variant was saved as a separate XML file for use in subsequent experiments.

This approach can be easily extended to generate a much larger set of morphologies, enabling applications such as domain randomization and improving generalization across varying body structures.

4.4 Reward Function

The objective of the reinforcement learning agent is to maximize the cumulative discounted return over an episode horizon of $T = 1000$ timesteps.

To guide the agent toward learning robust and efficient locomotion across diverse morphologies, we define a multi-objective reward function R_t for each time step t . The design follows the standard MuJoCo paradigm, balancing the trade-off between task performance (velocity), survival (stability), and energy efficiency (control effort).

The total reward R is formulated as a weighted linear combination of three distinct components:

$$R = w_h \cdot r_{healthy} + w_f \cdot r_{forward} - w_c \cdot r_{control} \quad (4.1)$$

where the coefficients are defined as $w_h = 1.0$, $w_f = 1.0$, and $w_c = 0.5$. These terms are defined as follows:

1. **Healthy Bonus ($r_{healthy}$):** To discourage the agent from adopting unstable configurations that lead to early episode termination, a constant reward is granted for every timestep the agent remains in a defined "healthy" state.

$$r_{healthy} = \begin{cases} 1.0 & \text{if is_healthy}(m, d_1) \\ 0.0 & \text{otherwise} \end{cases} \quad (4.2)$$

The `is_healthy` criterion checks if the torso height remains within a valid range.

2. **Forward Velocity ($r_{forward}$):** This term represents the primary task objective, maximizing displacement along the x-axis. In the MJX implementation, the velocity is calculated via finite differences of the root position q_{pos} over the simulation step Δt :

$$r_{forward} = \frac{q_{pos,x}(t + \Delta t) - q_{pos,x}(t)}{\Delta t} \quad (4.3)$$

3. **Control Penalty ($r_{control}$):** To promote the emergence of natural-looking gaits and penalize excessive energy expenditure, we apply a penalty on the magnitude of the control signals u . This prevents high-frequency oscillations in the actuators:

$$r_{control} = \sum_{j=1}^n u_j^2 \quad (4.4)$$

By penalizing control effort while rewarding forward progress, the agent is incentivized to find an energy efficient gait.

4.5 Training Procedure

The agents were trained using evolutionary strategy approach based on the method proposed by Salimans et al. (2017). In this framework, policy parameters are optimized through a population-based search rather than gradient-based reinforcement learning.

In each generation, a population of parameter perturbations was sampled and evaluated in the environment. The performance of each individual was measured by the cumulative reward obtained during an episode, and these evaluations were used to update the policy parameters.

A population size of 1024 individuals was used for each environment, for a total of 4096. Each candidate solution is evaluated across all training morphologies. The resulting returns are then averaged across environments. To mitigate discrepancies in reward scale across different morphologies, rewards were normalized on a per-environment basis before averaging. Specifically, for each morphology, the rewards obtained by the population were rank-normalized, ensuring that only the relative performance of individuals within that environment was considered. This normalization step is important because certain morphologies, such as those with longer limbs, can inherently achieve higher absolute rewards due to their ability to cover more distance. Without normalization, these easy environments would disproportionately influence the gradient. By applying rank normalization prior to averaging across environments, each morphology contributes equally to the overall objective, thereby promoting learning of policies that generalize across different body configurations.

Both the learning rate and the standard deviation of the parameter perturbations were set with the initial value to 0.1. An exponential decay schedule for both the learning rate and the mutation noise, with a decay rate of 0.999 was used, this ensures broad exploration of the plasticity space in early generations while allowing for fine-tuning as the population converges. These hyperparameters were found by trial and error. The optimization process was run over 500 generations. All parameters were initialized to zero at the start of training. Each evaluation episode was run for a fixed horizon of 1000 timesteps.

Training one agent takes approximately 2 hours on a Nvidia A100 GPU.

5 Experimental Results and Discussion

5.1 Training Performance

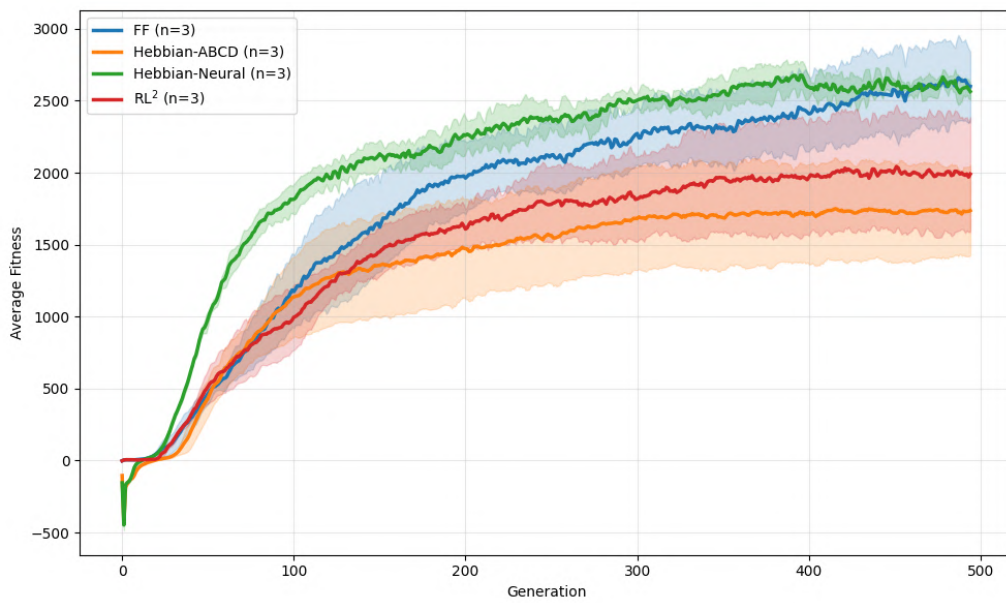


Figure 5.1: Training curves for the robot locomotion task. The graph shows the average and standard deviation of the reward on the training set across three trials for each model.

Figure ?? shows the training performance of all evaluated models across generations. Among the compared methods, the NeuralHL model demonstrates the highest sample efficiency, achieving high fitness values substantially faster than other methods. NeuralHL rapidly increases in performance during the early stages of training and reaches a stable reward level near 2500 within approximately the first half of the training process.

The feed-forward baseline eventually converges to a similar final fitness level as NeuralHL, but requires considerably more generations to do so. This suggests that although the baseline possesses sufficient representational capacity to learn effective locomotion policies for the training morphologies, it requires more optimization steps to discover these solutions. In contrast, the NeuralHL approach appears to benefit from adaptive plasticity dynamics, enabling faster acquisition of successful control strategies and improved learning efficiency.

The standard Hebbian-ABCD model has overall the slowest training dynamics. Its reward curve plateaus below 2000 and shows only limited improvement across generations. This behavior may indicate that the fixed ABCD parametrization lacks sufficient flexibility to support effective adaptation in the locomotion task. Compared to the NeuralHL model, which uses a more expressive neural parametrization of the plasticity rule, the simpler Hebbian formulation may restrict the diversity of behaviours that can emerge during optimization.

The RL^2 model demonstrates intermediate performance, with a plateau a little bit above Hebbian-ABCD. It shows relatively stable and consistent learning dynamics throughout training, steadily improving to a final reward near 2000.

Interestingly, despite achieving comparable final training rewards, the NeuralHL and feed-forward baseline models exhibit markedly different learning dynamics. NeuralHL reaches high-performance solutions rapidly, whereas the baseline improves more gradually over time. This difference suggests that local plasticity mechanisms may not necessarily increase the maximum achievable reward in the training environment, but instead improve the speed and efficiency with which effective policies are acquired.

Overall, these results indicate that increasing the expressivity of local plasticity rules substantially improves training efficiency and optimization dynamics in the robot locomotion task.

5.2 Generalization to Unseen Morphologies

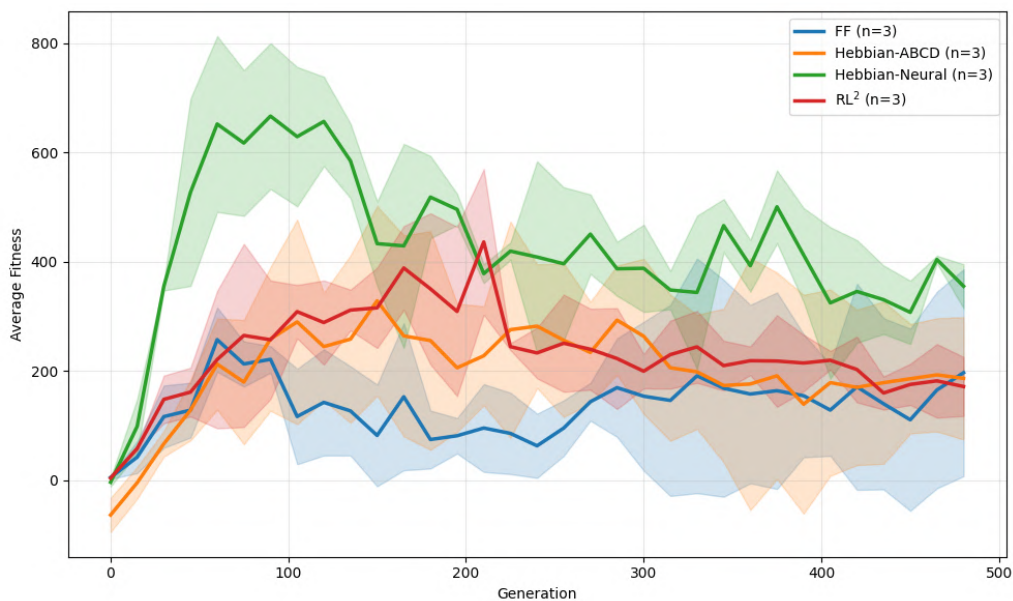


Figure 5.2: Evaluation curves on unseen robot morphologies. The graph shows the average and standard deviation of the reward on the test set across three trials for each model.

Figure 5.2 shows the performance of all models on the 11 unseen evaluation morphologies. In contrast to the training environments, all methods achieve substantially lower absolute fitness values on the test set, indicating a significant gap between training and generalization performance. This suggests that all evaluated approaches exhibit some degree of overfitting to the limited set of training morphologies.

Among the compared models, the Hebbian-Neural approach demonstrates the strongest generalization performance. The model rapidly reaches a peak evaluation reward above 600 within approximately the first 100 generations, substantially outperforming the remaining approaches during early training. This behaviour suggests that the neural parametrization of local plasticity initially enables the emergence of highly adaptable control strategies capable of transferring to previously unseen body configurations.

However, despite this strong early performance, the evaluation reward of Hebbian-Neural gradually declines throughout later generations. While the model continues improving on the training environments, its ability to generalize to unseen morphologies deteriorates over time. This divergence between training and evaluation performance suggests overfitting to the training morphologies. The continued optimization drives the controller toward increasingly specialized locomotion strategies that exploit the specific structural properties of the training morphologies, thereby reducing flexibility under novel limb configurations.

The feed-forward baseline exhibits weak and unstable generalization performance. Although the model achieves moderate evaluation rewards during early training, its performance remains consistently below the peak performance of other methods. Since the feed-forward controller lacks any mechanism for within-episode adaptation, it must rely entirely on policies learned during training. The poor transfer performance therefore suggests that fixed policies alone are insufficient to robustly generalize across morphological variations in the locomotion task.

Although RL² does not achieve the same peak performance as Hebbian-Neural, its performance degrades less severely during later generations.

The Hebbian-ABCD model achieves lower peak performance than Hebbian-Neural but exhibits comparatively stable evaluation dynamics across generations. While its absolute rewards remain moderate, the model does not experience the pronounced decline observed in Hebbian-Neural. One possible explanation is that the simpler ABCD plasticity parametrization limits the capacity of the model to over-specialize to the training environments. In this sense, the reduced expressivity of the plasticity rule may act as an implicit regularizer that stabilizes generalization performance.

As in the training curves, RL² and Hebbian-ABCD have very similar dynamics, with RL² being

overall slightly better.

The evaluation results indicate that expressive adaptive plasticity mechanisms can substantially improve generalization to unseen morphologies, but may also increase susceptibility to overfitting when training diversity is limited. In contrast, simpler or less expressive adaptive mechanisms appear to produce more stable, though less optimal, generalization behaviour.

5.3 The Mean Episodic Reward (MER)

The Mean Episodic Reward (MER) was used as a performance metric and was computed as the average cumulative reward across all episodes:

$$\text{MER} = \frac{1}{N_{\text{total}}} \sum_{i=1}^{N_{\text{total}}} \left(\sum_{t=0}^H r_{i,t} \right), \quad (5.1)$$

where N_{total} denotes the total number of evaluated episodes, H is the episode horizon, and $r_{i,t}$ is the reward obtained at time step t in episode i .

Table 5.1: Mean Episodic Reward (MER) for each method, averaged over all generations and runs.

Method	Training MER	Evaluation MER
FF	1815.36	135.23
Hebbian-ABCD	1352.01	201.05
Hebbian-Neural	2092.96	424.78
RL ²	1503.17	232.13

As shown in Table 5.1, the Hebbian-Neural model achieved the highest Mean Episodic Reward in both the training and evaluation environments, with MER values of 2092.96 and 424.78, respectively. This indicates that the neural Hebbian plasticity mechanism provided the strongest overall performance among the evaluated methods and also generalized best to the evaluation environments.

On the training environments, the feed-forward baseline obtained the second-highest MER of 1815.36, outperforming both RL² and Hebbian-ABCD. However, this performance did not transfer to the evaluation environments, where the feed-forward model achieved the lowest MER of 135.23. This large drop indicates that the feed-forward baseline strongly overfit to the training environments. Since the feed-forward baseline is the only method without the capacity to adapt within an episode, these results confirm that within-episode adaptation is an important capability for this challenge. In contrast, both plastic approaches and RL² obtained higher evaluation rewards than the feed-forward baseline, suggesting better adaptation or generalization to unseen environ-

ments. Overall, these results show that Hebbian-Neural was the most effective method across both settings, while Hebbian-ABCD performed worse during training but showed improved evaluation performance relative to the feed-forward baseline.

5.4 Limitations and Threats to Validity

Several limitations should be considered when interpreting the results of this study. The most important limitation concerns the small number of training morphologies. Only four morphologies were used during optimization, providing limited coverage of the possible body configurations that the controllers may encounter. The main reason for that is that every morphology must be compiled separately. This introduces a significant overhead when scaling to larger morphology sets. Ideally, all morphologies would share the same compiled computation graph avoiding the need for recompilation. This should be possible within the JAX framework, but might require a lot of additional engineering effort. As a result of the small training set, the models learn morphology-specific locomotion strategies rather than more generally transferable control principles. This limitation results in the substantial gap between training and evaluation performance observed across all methods, indicating varying degrees of overfitting to the training environments. The test score starts to fall even as the training score keeps improving, which is also a textbook indication of overfitting.

6 Software Architecture Aspects

6.1 Overview

Software architecture is concerned with the high-level structure of software systems and the design decisions that determine system qualities Bass et al. (2003). It studies how a system is decomposed into components, how these components interact, how data flows through the system, and how architectural decisions affect important quality attributes. These quality attributes might include for example: performance efficiency, scalability, maintainability, extensibility, reproducibility, and resource usage. In contrast to lower-level implementation details, software architecture focuses on design decisions that shape the overall behaviour and future evolution of the system. Such decisions are often difficult to reverse and therefore strongly influence whether a system can satisfy its intended goals.

As a research field, software architecture aims to understand how architectural choices influence system qualities under specific constraints. A central theme is the analysis of trade-offs. In many systems, improving one quality attribute may negatively affect another. Architecture research therefore asks why a particular structure is appropriate, which alternatives exist, and under which constraints one design should be preferred over another.

6.2 MLOps

The transition of machine learning (ML) models from isolated experimental notebooks to robust production environments requires structured operational workflows. Machine Learning Operations (MLOps) has emerged as the standard paradigm to unify machine learning system development and system operations. MLOps standardizes the automation of data pipelines, model training, continuous integration and continuous deployment (CI/CD), and performance monitoring to ensure reproducibility and mitigate data drift over time Kreuzberger et al. (2023).

Architectural questions are important in this context. In traditional software systems, the behaviour of the system is mostly specified directly through code. In machine learning systems, how-

ever, part of the system behaviour is learned from data or interaction with an environment. As a result, the architecture must manage not only source code, but also models, parameters, training pipelines, datasets or simulators, evaluation protocols, hardware resources, and experiment configurations. This makes machine learning systems particularly dependent on reproducibility, observability, scalability, and efficient resource management.

The central software architecture concern in this thesis is therefore the trade-off between adaptability and system complexity. More expressive adaptive mechanisms may improve learning and generalization, but they also require additional computation and may become harder to scale. Simpler architectures may be more efficient and easier to deploy, but may lack the flexibility needed for changing environments. Evaluating these trade-offs is important for understanding when plastic neural networks are useful from an engineering perspective.

6.3 The End-to-End Machine Learning Lifecycle

Many machine learning projects never actually make it past the proof-of-concept stage Kreuzberger et al. (2023) and therefore widespread deployment of ML in real-world industrial systems remains limited. In many practical applications, industries continue to rely on established approaches such as rule-based systems or PID controllers, which are easier to validate, interpret, monitor, and deploy reliably in production environments. To close this gap and successfully transition complex models from the lab to the field, a structured, end-to-end operational pipeline is essential; the core stages of this pipeline are detailed below.

Data Collection Data collection refers to the process of gathering relevant data from available sources. This step provides the foundation for the machine learning system, since the quality, diversity, and representativeness of the collected data directly influence the performance of the trained model. This step might also include filtering, preprocessing of the data, gathering metadata, and any other process to assemble a complete and usable dataset.

Model Training During model training, the machine learning algorithm learns patterns from the collected data. The model adjusts its internal parameters to minimize prediction errors and improve its ability to generalize to unseen examples.

Evaluation / Validation Model evaluation and validation are used to assess whether the trained model performs reliably on data that was not used during training. This step helps identify overfit-

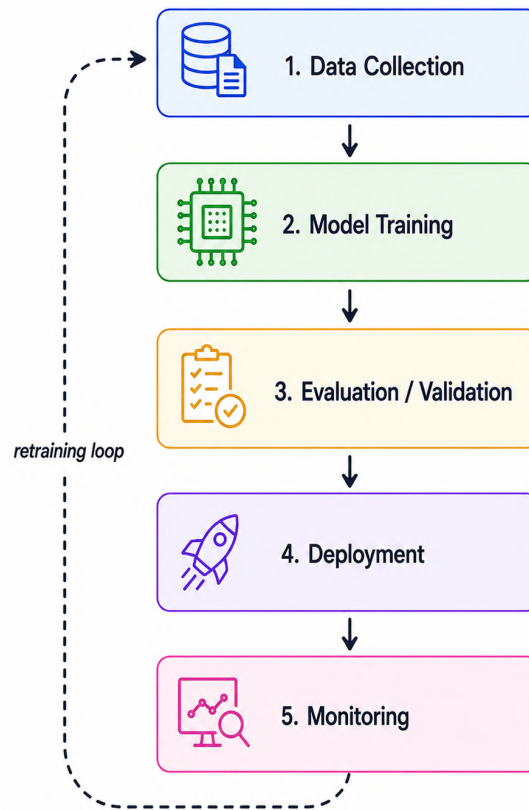


Figure 6.1: Overview of the machine learning lifecycle

ting, poor generalization, or insufficient performance before the model is deployed.

Deployment Deployment is the process of integrating the trained and validated model into a production environment. At this stage, the model becomes available for real-world use, for example through a GUI or an API.

Monitoring Monitoring involves continuously observing the deployed model’s performance and behaviour over time. This step is necessary because data distributions, user behaviour, or system conditions may change after deployment, potentially reducing model performance.

Retraining loop The retraining loop represents the iterative improvement of the machine learning system after deployment. Feedback and newly collected data from the monitoring stage can be used to update or retrain the model, allowing the system to adapt to changing data distributions and maintain reliable performance over time. Industrial recommender systems, which are some of the most widely deployed machine learning systems today, often require very frequent retraining due to data drift. The performance of deployed models tends to degrade over time due to item catalogue updates and user preference changes. For example, popular food delivery ap-

plication Grubhub reported that “updates at a daily cadence, if not more frequent, are required, to use machine learning in production for our recommender use-case” Egg (2021), with daily retraining substantially outperforming weekly retraining. Such frequent retraining imposes a significant financial cost, and requires a highly automated efficient pipeline, which is the object of the field of MLOps. This differs completely to academic settings where models are only trained once as a proof of concept and never deployed.

6.4 Architectural Implications of Continuous Adaptation

Efficient continual learning remains an open problem. Current systems still struggle with challenges such as catastrophic forgetting, and practical machine learning pipelines therefore rely on periodic retraining instead of true continuous adaptation. The highly automated infrastructure used for frequent retraining can, taken as a whole, be viewed as a basic form of continual learning. It corresponds to the fully plastic end of the spectrum as we have defined it in Section 2.7. However, this system is extremely inefficient, because retraining on a daily basis requires repeatedly computing all previously data in each training iteration to incorporate only a small amount of additional new of data.

Because full retraining is inefficient, future machine learning systems will likely rely more on methods that update models without repeatedly recomputing all previous data. This shift will also change the machine learning lifecycle, because adaptation becomes a continuous part of the system rather than a separate retraining step. While reducing the frequency of retraining, continuous adaptation also introducing challenges in other parts of the pipeline.

Continuous adaptation introduces several MLOps challenges. Since the model can change during operation, its behaviour becomes less predictable and harder to reproduce. This complicates debugging, testing, and reliability analysis, especially compared to static models. It also increases monitoring requirements, because the system must observe not only task performance, but also whether the adaptation process itself remains stable. Finally, adaptive systems may degrade if they learn from noisy, biased, or corrupted inputs, so practical architectures require mechanisms to control or limit harmful adaptation.

Overall, continuously adaptive neural networks transform machine learning systems from static predictive models into dynamic, evolving systems. Machine learning systems are already difficult to test and validate compared to traditional software, because their behaviour is often probabilistic. Continuous adaptation increases this difficulty further.

6.5 Rethinking the Foundations of MLOps for Continual Reinforcement Learning

The paper *Rethinking the Foundations for Continual Reinforcement Learning* Elelimy et al. (2025) argues that many ideas from traditional reinforcement learning are not well suited for continual reinforcement learning. In traditional RL, the goal is usually to train an agent until it finds an optimal or near-optimal policy. After that, learning is considered finished.

The authors identify four traditional foundations, or “pillars”, that define standard RL. First, RL is usually formulated using Markov decision processes, which often assume a stable environment and repeatable states. This is problematic in real-world continual learning, where the environment may change and the agent may never encounter exactly the same situation again. Second, traditional RL focuses on fixed artifacts, such as an optimal policy or value function. In continual RL, however, the important object is not a final fixed model, but the ongoing adaptive behaviour of the agent. Third, RL is commonly evaluated using the expected sum of rewards. This assumes that performance can be measured over repeated episodes, but continual environments may not allow clean resets or repeated trials. Fourth, many RL benchmarks are episodic, meaning that they restart after each episode. Such benchmarks do not fully represent never-ending learning problems.

These four pillars are useful for connecting continual RL to MLOps. Standard MLOps pipelines are also built around a relatively fixed model artifact: a model is trained, evaluated, deployed, monitored, and later retrained if necessary. This lifecycle fits traditional ML, where learning is mostly separated from deployment. However, continual RL challenges this separation. If the agent keeps adapting during deployment, then training, evaluation, deployment, and monitoring can no longer be treated as independent stages. Instead, the whole lifecycle must be reconsidered around an adaptive process rather than a static model.

Below, we list the stages of the machine learning lifecycle and explain how they must be modified.

Data Collection This aspect remains largely unchanged, although since the continues to adapt during deployment, the data preprocessing, filtering and other steps in the data collection pipeline might have to also run continuously.

Model Training Since part of the adaptation can occur during deployment, full offline retraining is required less frequently and therefore becomes a smaller cost factor. However, this reduction comes at the expense of increased complexity and cost in other lifecycle stages, especially deploy-

ment, monitoring, validation, and safety control. As a result, plasticity does not remove operational costs entirely, but shifts them from retraining pipelines toward runtime adaptation management. This trade-off is mainly worth it when frequent retraining is expensive, especially when such retraining is required to respond to rapidly drifting data distributions.

Evaluation / Validation In continuously adaptive systems, validation can no longer be treated as a one-time activity because the deployed model continues to change throughout its lifetime. Evaluation must therefore become an ongoing process integrated into the deployment environment. This may involve continuously tracking performance metrics such as average per step reward. Another solution could be to periodically evaluating model snapshots on validation datasets.

Deployment When deploying a model that changes over time, periodic snapshots of the model state become important for reproducibility. If an error occurs, saved snapshots can help reconstruct the model state that produced the failure, making debugging more feasible. In addition, snapshots provide a rollback mechanism: if a harmful adaption causes the model parameters to move into an unsafe state, the system can be reverted to a previously validated version. Thus, model deployment in plastic systems must include not only release management, but also state tracking, checkpointing, and recovery mechanisms.

Another difference here is in distributed or edge deployments: if the same plastic model is deployed to multiple users or devices, each instance will receive different inputs and therefore the parameters will diverge over time. As a result, the deployed system can no longer be treated as a single fixed model version. Instead, the architecture must account for divergence between model instances, including how local adaptations are logged, synchronized, constrained, or rolled back.

Monitoring Additional metrics need to be monitored here. In addition to metrics such as latency and resource usage, the system also needs to monitor properties of the adaptation process itself, such as the rate of change of model parameters, in order to flag potentially disruptive adaptation. To some extent, the Monitoring phase therefore merges with the Evaluation / Validation phase, since the model is not only observed for operational failures, but also continuously assessed for behavioural changes. This might result in two phases that use the same metrics: pre-deployment monitoring, which evaluates whether the initial model is stable enough to release, and post-deployment monitoring, which observes whether continued adaptation remains safe and useful during operation.

Retraining loop The retraining loop does not disappear, but it becomes less central. In traditional MLOps, retraining is the main way in which the model adapts to new data. In plastic systems, part of this adaptation happens directly during deployment, so full offline retraining may be needed less frequently. However, retraining is still useful as a maintenance mechanism, for example to correct harmful adaptation, or produce a new stable model version. Thus, retraining shifts from being the primary adaptation mechanism to being a periodic correction and stabilization mechanism.

7 Conclusion and Future Work

7.1 Thesis Summary

This thesis investigated meta learning with Hebbian plasticity, first, a survey of the relevant literature was conducted to identify the current state of the field and its main application domains. Second, an experimental study evaluated whether meta-learned Hebbian plasticity improves adaptation to unseen robot morphologies. Third, the deployment implications of continuously adaptive systems were analysed from an MLOps perspective. Together, these contributions provide insights into both the learning capabilities and practical deployment challenges of plastic neural networks.

This survey has examined the landscape of meta-learning with Hebbian plasticity, a biologically inspired framework that leverages local, unsupervised, and online update mechanisms. Plastic neural networks demonstrate a unique capacity for rapid adaptation and efficient memory storage. As we have seen, these methods are particularly effective in challenging domains such as continual learning, few-shot learning, and reinforcement learning, where the ability to generalize across non-stationary environments is critical. Addressing the high meta-parameter costs and balancing expressive capacity with scalability is essential for tackling harder high-dimensional benchmarks. Looking forward, the synergy between Hebbian learning and neuromorphic hardware presents a promising path toward energy-efficient, low-latency artificial intelligence

7.2 Research Question Evaluation

The results obtained in this study allows us to answer our research questions:

7.2.1 RQ1: Does meta-learned plastic networks outperform static networks in adapting to unseen environments?

The results provide evidence in support of this hypothesis. The feed-forward baseline exhibited the poorest performance transfer to unseen morphologies. In contrast Hebbian-Neural, Hebbian-ABCD, achieved higher evaluation rewards. These findings suggest that Hebbian plasticity provides

a clear advantage when generalizing to previously unseen environments.

7.2.2 RQ2: Does increasing the expressivity of the local plasticity rule improve generalization?

The results support this hypothesis. Across all performance metrics, the Hebbian-Neural model consistently outperformed the simpler Hebbian-ABCD formulation. The neural parametrization achieved higher training rewards, superior sample efficiency, stronger initial generalization performance, and the highest Mean Episodic Reward on both the training and evaluation morphologies.

These findings suggest that increasing the expressivity of local plasticity rules enables the emergence of more effective adaptive behaviours.

Overall, the results indicate that both research questions can be answered affirmatively. Adaptive mechanisms outperform static policies when transferring to unseen morphologies, and more expressive Hebbian plasticity rules provide additional gains in learning efficiency and generalization performance.

7.2.3 RQ3: How does plasticity alter the MLOps lifecycle compared to traditional external retraining pipelines?

Plasticity changes the role of the MLOps lifecycle by relocating adaptation from offline retraining pipelines to the deployed model itself. The reduction in retraining effort is accompanied by increased architectural complexity during deployment. To some extent the MLOps pipeline also has to be running continually to monitor performance, trigger rollbacks and retraining. The primary engineering challenge shifts from managing large-scale retraining infrastructure to managing and controlling adaptation at runtime.

7.2.4 RQ4: What architectural patterns and safety mechanisms are required to monitor, validate, and control model behaviour during continuous deployment?

Continuously adaptive systems require architectural mechanisms that make the adaptation process observable, controllable, and reversible. Traditional monitoring of performance metrics alone is insufficient because model behaviour can change even when the deployed software remains unchanged. Effective deployment therefore requires monitoring adaptation dynamics, continuous validation during operation, and safety mechanisms that can detect and mitigate harmful updates. Architectural patterns such as adaptation constraints, anomaly detection, rollback mechanisms,

and fallback models are necessary to maintain reliability. The key requirement is not preventing adaptation, but ensuring that adaptation remains bounded, observable, and recoverable throughout the system’s lifetime.

7.3 Key Findings

The results presented demonstrate clear differences in both learning dynamics and generalization performance between the evaluated adaptation mechanisms. Among all methods, the Hebbian-Neural model consistently achieved the strongest overall performance. It exhibited the highest sample efficiency during training, reached high fitness values substantially faster than the competing approaches, and achieved the highest Mean Episodic Reward in both the training and evaluation environments.

The evaluation on unseen morphologies revealed that adaptive mechanisms generally transfer more effectively than fixed policies. While the feed-forward baseline achieved competitive performance on the training environments, it exhibited the largest drop in reward when evaluated on unseen morphologies, indicating substantial overfitting to the training set. In contrast, both Hebbian approaches and RL² achieved improved evaluation performance, suggesting that within-episode adaptation contributes to robustness under morphology shifts.

The Hebbian-Neural model demonstrated particularly strong initial generalization, substantially outperforming all other methods during the early stages of training. However, its evaluation performance gradually declined as training progressed, indicating that increased plasticity expressivity may also increase susceptibility to over-specialization when training diversity is limited. Interestingly, the simpler Hebbian-ABCD model and RL² exhibited more stable evaluation behaviour, although at a lower performance level.

This study differs from much of the existing literature in two important respects. Many previous studies on Hebbian plasticity evaluate adaptation by training agents in a single environment or morphology and subsequently testing them under modified conditions. In contrast, the models in this work were optimized across a set of training morphologies, requiring them to learn control strategies that perform well over a distribution of body configurations rather than a single fixed morphology. This setting provides a more demanding test of adaptation and generalization and may better reflect real-world scenarios where agents encounter continuous variation.

Furthermore, Hebbian plasticity is often compared only against static feed-forward architectures, making it difficult to determine whether its benefits arise specifically from plasticity or more generally from the presence of adaptive mechanisms. By additionally comparing Hebbian plasticity to

RL², an established meta-learning approach, this study provides a more direct evaluation of how local plasticity-based adaptation performs relative to alternative adaptive learning mechanisms. The results suggest that Hebbian plasticity is competitive.

Taken together, these findings suggest that increasing the expressivity of local Hebbian plasticity rules can substantially improve learning efficiency, adaptation capability, and overall performance. At the same time, the persistent gap between training and evaluation rewards across all methods highlights the challenge of achieving robust generalization under limited morphology diversity. Future experiments involving larger and more diverse morphology distributions will be necessary to determine the extent to which the observed advantages of neural Hebbian plasticity generalize beyond the current experimental setting.

7.4 Future Work

Addressing the high meta-parameter costs and balancing expressive capacity with scalability is essential for tackling harder high-dimensional benchmarks. Looking forward, the synergy between Hebbian learning and neuromorphic hardware presents a promising path toward energy-efficient, low-latency artificial intelligence

7.4.1 Experimental Extensions

Several directions for future work emerge from these findings. A natural extension would be to train the models on a substantially larger and more diverse set of morphologies. It would be valuable to explore whether the overfitting observed, dissipates once the training set becomes larger and it would provide a stronger test of morphology-generalizing adaptation. In addition, the morphological variations considered in this study are restricted to differences in limb length, while the overall topology of the robot remains fixed. Consequently, the evaluation setup captures only a relatively narrow subset of possible structural variations. More substantial topology changes, such as variations in the number of limbs or joint connectivity as in Gupta et al. (2021), would present much greater adaptation challenges.

Another promising direction would involve evaluating the proposed approaches in additional reinforcement learning environments beyond robot locomotion. This would help determine whether the observed advantages of neural Hebbian plasticity generalize across tasks with different adaptation requirements and dynamics. Overall, while the present results demonstrate promising advantages for expressive local plasticity mechanisms, further experiments are necessary to fully evaluate

their robustness, scalability, and generalization capabilities.

7.4.2 Neuromorphic and Spiking Systems

Spiking Neural Networks (SNNs) differ from conventional Artificial Neural Networks (ANNs) in that information is transmitted through discrete spikes over time rather than continuous-valued activations. As a result, computation is event-driven: neurons only communicate and perform computation when an input spike occurs, rather than updating continuously at every timestep. Additionally, whereas in conventional ANNs all neurons are evaluated during each forward pass through matrix multiplications, even when many activations are zero, SNNs can exploit sparse computation, meaning that only neurons that emit spikes and their associated synapses participate in the computation. Together, these two properties of event-driven execution and sparse computation significantly reduce energy consumption and computational cost.

A major difficulty, holding back the application of SNNs is that they are not differentiable. Therefore identifying effective learning algorithms is an important question in that field, and the methods most often used today try to awkwardly fit SNNs into a framework made for ANN, by using surrogate gradients Neftci et al. (2019) or ANN to SNN conversion Rueckauer et al. (2017). However the methods used in this thesis, which leverage Evolutionary Strategy and Hebbian learning seem ideally suited for SNNs, since both don't depend on gradient descent. With SNNs the hebbian learning rule can be naturally implemented and parametrised as Spike-Timing Dependent Plasticity (STDP). There are opportunities to apply meta-learning of plasticity rules within that area (see Confavreux et al. (2020) and Jordan et al. (2021) for example), but research is still at a very early stage.

SNNs are also particularly interesting because they can be run efficiently on neuromorphic hardware. Neuromorphic hardware is an emerging paradigm that offers much better energy efficiency and lower latency compared to current GPU-based computing; two key bottlenecks in modern machine learning. Furthermore, some neuromorphic systems implement Hebbian learning directly at the hardware level Eryilmaz et al. (2014), making them ideally suited for these types of learning algorithms.

Appendix A: Reproduction of the Original RL² Paper

A.1 Abstract

In order to verify that the RL² implementation used in Chapter 4 was correct we replicate here the experiments described in the original paper by Duan et al. (2016).

Deep reinforcement learning has demonstrated the ability to automatically acquire sophisticated behaviours; however, such learning typically requires extensive training. The RL² method was proposed in order to overcome this limitation using the concept of meta-learning reinforcement learning algorithms applied to recurrent neural networks. This method was independently proposed around the same time by Duan et al. (2016) under the name of RL² Wang et al. (2016) and under the title 'Learning to Reinforcement Learn'. The idea is to embed a "fast" reinforcement learning algorithm within the weights of a recurrent neural network (RNN). And to leverage a "slow" RL process to train the RNN on data from diverse tasks, enabling rapid adaptation to new, unseen environments. In this reproduction study, we reimplement and evaluate the RL² framework to verify its effectiveness and reproducibility. Specifically, we replicate experiments conducted on both small-scale problems, such as multi-armed bandits and finite Markov Decision Processes (MDPs), and a navigation task. Our findings align closely with the original results, confirming RL²'s ability to achieve near-optimal performance in previously unseen tasks and highlighting its robustness and scalability. This work confirms the reproducibility of RL² and further validates the feasibility of learning reinforcement learning algorithms directly through recurrent neural networks.

A.2 Introduction

Deep reinforcement learning has achieved impressive results but often relies heavily on massive amounts of training data, limiting its practical applicability and efficiency. Human-designed domain specific solutions do not scale easily, as they depend on manual effort. An alternative is

meta-learning, which enables algorithms to acquire useful biases automatically through experience. Such methods, including RL², offer a scalable path toward systems that rapidly generalize to new tasks.

A.3 Method

A.3.1 Reinforcement Learning Formalism

An RL algorithm learns a policy to take actions in a Markov Decision Process (MDP). An MDP is defined by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, P_0, R, \gamma, T \rangle$,

where:

- $\mathcal{S}$ is the set of states,
- $\mathcal{A}$ is the set of actions,
- $P(s_{t+1} \mid s_t, a_t) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}^+$ is the probability of transitioning from state s_t to s_{t+1} after taking action a_t ,
- $P_0(s_0) : \mathcal{S} \rightarrow \mathbb{R}^+$ is a distribution over initial states,
- $R(s_t, a_t) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function,
- $\gamma \in [0, 1]$ is the discount factor,
- T is the episode horizon.

A policy is a function $\pi(a \mid s) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^+$ that maps states to action probabilities.

The interaction of the policy with the MDP takes place in episodes, which start from an initial state sampled from P_0 , followed by T transitions. At each step, an action is sampled from the policy π , and the next state from the dynamics P . The reward for each transition is given by $r_t = R(s_t, a_t)$.

The objective of reinforcement learning is to learn a policy that maximizes the expected discounted return within an episode:

$$J(\pi) = \mathbb{E}_{\tau \sim P(\tau)} \left[\sum_{t=0}^T \gamma^t r_t \right],$$

where τ denotes a trajectory of states, actions, and rewards.

A.3.2 Meta-RL

In meta reinforcement learning the goal is not to learn a policy but to learn a reinforcement algorithm that will output a policy. Two different learning systems exist in a layered structure. The first, the outer loop, is responsible for learning to learn. The second, the inner loop, is responsible for adapting to new tasks. The outer loop slowly tunes and modifies the parameters of the inner loop algorithm. It is trained on a distribution of tasks, these tasks differ but must share some common characteristics to allow for useful prior knowledge extraction. The outer loop is trained on each new task for one or multiple episodes which together form a trial. Its goal is not to maximise the reward sum received by the agent in a single episode but rather over an entire trial. This incentivises the creation of a fast-learning inner loop algorithm. The outer loop, is typically a standard backpropagation-based optimisation algorithm, while in the method presented here the inner layer resides within the recurrent weights of an recurrent neural network.

A.3.3 Explanation of the RL² method

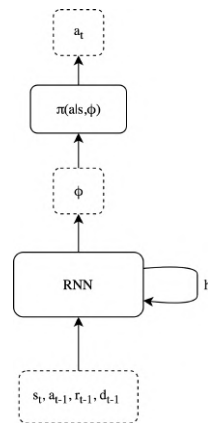


Figure A.1: Diagram illustrating the RL² method structure.

The current state s_t , the previous action a_{t-1} , the previous reward r_{t-1} , and the previous done flag d_{t-1} are concatenated and passed to the RNN, which, conditioned on its hidden state h_t , outputs a context vector. This context vector then conditions the policy MLP, which in turn outputs the action a_t .

Essentially, the RNN receives the same information as a standard RL algorithm—namely, the State-Action-Reward tuple and therefore will be able to improve its performance at a given task. In this setup, the RNN acts as the fast inner-loop learning system, fine-tuned to efficiently learn tasks sampled from the task distribution. A major advantage meta-RL is that because the learning process responsible for adapting to new tasks is entirely contained inside the inner loop, the agent can still improve its performance on a new task during run time when the agent has been

deployed in the real world and its weights are frozen. For this reason this approach can be useful in non-stationary environments as it helps the agent quickly adapt to changes in the environment dynamics.

A.4 Experiments

A.4.1 Multi-Armed Bandits

Multi-armed bandits are a simplified subset of MDPs where the environment is stateless. Specifically, the problem consists of k possible actions (arms), each associated with an unknown reward distribution. At each timestep, the agent selects one of the arms, and receives a reward drawn from its associated distribution. In our experiments, the reward distribution for each arm is a Bernoulli distribution characterized by a parameter θ . Randomly sampled for each arm from the uniform distribution on $[0, 1]$.

The objective of the multi-armed bandit problem is to maximize the total reward accumulated over a fixed number of timesteps. The main challenge is effectively balancing exploration (to accurately estimate each arm’s reward distribution) and exploitation (choosing the arm believed to yield the highest reward).

We evaluate the RL² method. If successful, the trained RNN policy should achieve performance comparable to standard baseline strategies

Table A.1: Average total reward over 3000 episodes for different episode length and number of arms n, k . Note: The original paper also considers the $n = 500$ case, which is not included here due to computational resource constraints.

n, k	Random	Gittins	TS	OTS	UCB1	ϵ -Greedy	Greedy	RL ²
$n=10, k=5$	5.0	6.1	5.8	6.1	6.6	5.7	6.5	6.6
$n=10, k=10$	5.0	6.1	5.6	5.7	6.7	5.7	6.6	6.7
$n=10, k=50$	5.0	6.1	5.2	5.2	6.7	5.8	6.6	6.8
$n=100, k=5$	49.7	76.8	74.4	74.7	77.8	74.0	74.5	77.1
$n=100, k=10$	49.7	82.0	76.8	77.7	83.2	77.0	77.6	82.0
$n=100, k=50$	49.9	83.4	64.4	65.2	84.7	78.1	78.0	84.0

Note: The original paper also considers the $n = 500$ case, which is not included here due to computational resource constraints.

A.4.2 Tabular MDPs

While the bandit problem is a natural and simple setting to evaluate whether a policy can effectively balance exploration and exploitation, it lacks the sequential structure typical of full MDPs.

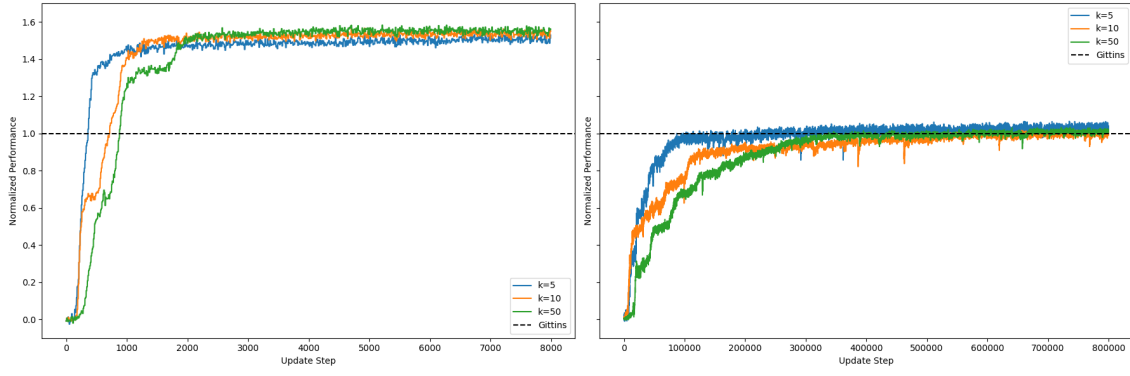


Figure A.2: RL2 learning curves for multi-armed bandits. Performance is normalized such that Gittins index scores 1, and random policy scores 0. $n=10$ (left) and $n=100$ (right)

To better capture the complexities of reinforcement learning, we conduct additional experiments on randomly generated tabular MDPs. These environments have a finite number of states and actions, with transition dynamics explicitly defined by a transition table. In this setting, we compare the performance of our approach against several established baseline methods.

Table A.2: Average total reward over 1000 episodes for different episode length T .

T	Random	PSRL	OPSRL	UCRL2	BEB	ϵ -Greedy	Greedy	RL ²
$T=5$	4.9	5.3	5.3	5.2	5.2	5.1	5.0	5.2
$T=10$	9.8	10.7	10.8	11.3	11.3	10.3	10.5	10.4
$T=20$	20	22.6	23.1	24.3	24.5	21.7	21.5	22.2
$T=35$	35	42	43.4	46.8	47.1	40.1	39.6	40.1
$T=50$	50	63.4	66	71.1	71.7	59.4	58.2	57.9

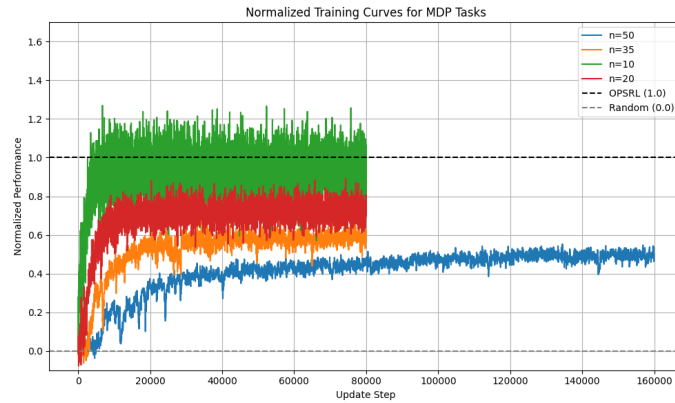


Figure A.3: learning curves for MDP. Performance is normalized such that OPSRL index scores 1, and random policy scores 0

A.4.3 Maze

To evaluate the method on a more challenging task, we also tested it on a grid maze exploration environment proposed. The original paper uses vision based 3d environment, here we use simpli-

fied 2d version. The maze consists of a 9×9 grid, enclosed by walls, with additional walls placed in every other square along both dimensions. This creates a regular pattern of 16 wall squares, forming a grid-like structure, except for the centre square. The maze layout remains fixed throughout the experiment. At the beginning of each episode, a reward location is randomly assigned to one of the non-wall squares. When the agent reaches this location, it receives a reward and is immediately teleported to a random position within the maze. Each episode spans 200 time steps, during which the agent aims to collect as much reward as possible. Importantly, the reward location stays fixed for the duration of an episode but changes between episodes. Since the reward location is not visible to the agent, it can only identify it upon receiving the reward.

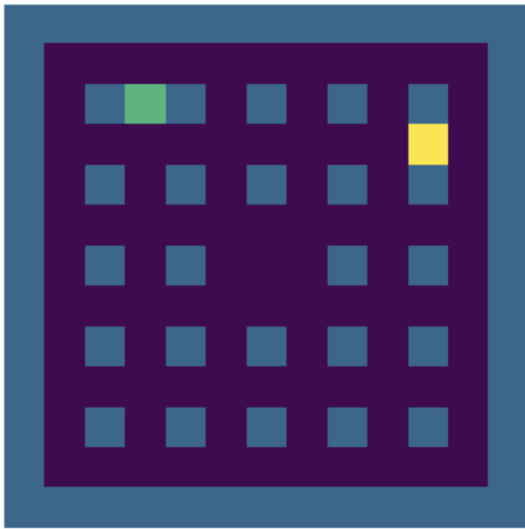


Figure A.4: Maze navigation task. Layout of the maze with example agent location (yellow) and reward location (green).

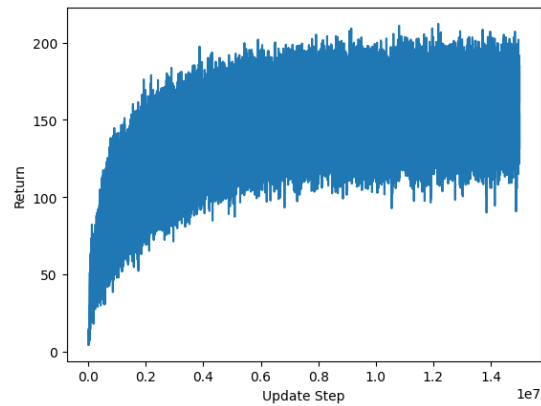


Figure A.5: Training curve for the maze exploration task. The agent learns a near-optimal policy.

A.5 Practical Recommendations

While reproducing this method, I encountered several implementation details that proved important for successful training. First, using a large batch size (e.g., 250000) significantly improved performance, smaller batch sizes consistently failed to produce good results in my experiments. Second, it is crucial to reset the RNN's hidden state between trials. However the hidden state should persist across episodes.

List of Figures

5.1	Training curves for the robot locomotion task. The graph shows the average and standard deviation of the reward on the training set across three trials for each model.	45
5.2	Evaluation curves on unseen robot morphologies. The graph shows the average and standard deviation of the reward on the test set across three trials for each model.	46
6.1	Overview of the machine learning lifecycle	53
A.1	Diagram illustrating the RL ² method structure.	67
A.2	RL2 learning curves for multi-armed bandits. Performance is normalized such that Gittins index scores 1, and random policy scores 0. n=10 (left) and n=100 (right) . . .	69
A.3	learning curves for MDP. Performance is normalized such that OPSRL index scores 1, and random policy scores 0	69
A.4	Maze navigation task. Layout of the maze with example agent location (yellow) and reward location (green).	70
A.5	Training curve for the maze exploration task. The agent learns a near-optimal policy.	70

List of Tables

- 2.1 Comparison of papers on plasticity in meta-learning. 28
- 3.1 Comparison of outer-loop optimized and inner-loop adaptive parameter count across models. 33
- 4.1 Comparison of simulation environments for evaluating plastic neural networks and universal controllers. 41
- 5.1 Mean Episodic Reward (MER) for each method, averaged over all generations and runs. 48
- A.1 Average total reward over 3000 episodes for different episode length and number of arms n, k . *Note:* The original paper also considers the $n = 500$ case, which is not included here due to computational resource constraints. 68
- A.2 Average total reward over 1000 episodes for different episode length T 69

Bibliography

- Ba, J., Hinton, G., Mnih, V., Leibo, J. Z., and Ionescu, C. (2016). Using fast weights to attend to the recent past.
- Bass, L., Clements, P., and Kazman, R. (2003). *Software Architecture in Practice*. SEI series in software engineering. Addison-Wesley.
- Bechtle, S., Molchanov, A., Chebotar, Y., Grefenstette, E., Righetti, L., Sukhatme, G., and Meier, F. (2021). Meta-learning via learned loss.
- Beck, J., Vuorio, R., Liu, E. Z., Xiong, Z., Zintgraf, L., Finn, C., and Whiteson, S. (2024). A survey of meta-reinforcement learning.
- Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.
- Ben-Iwhiwhu, E., Ladosz, P., Dick, J., Chen, W.-H., Pilly, P., and Soltoggio, A. (2020). Evolving in-born knowledge for fast adaptation in dynamic pomdp problems. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference, GECCO '20*, page 280–288. ACM.
- Bizzotto, E., Yaman, A., and Iacca, G. (2021). Promoting behavioral diversity via multi-objective/quality-diversity novelty producing synaptic plasticity. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 01–08.
- Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Katariya, Y., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs.
- Brown, T. H., Kairiss, E. W., and Keenan, C. L. (1990). Hebbian synapses: Biophysical mechanisms and algorithms. *Annual Review of Neuroscience*, 13(Volume 13, 1990):475–511.
- Coleman, O. J. and Blair, A. D. (2012). Evolving plastic neural networks for online learning: Review and future directions. In Thielscher, M. and Zhang, D., editors, *AI 2012: Advances in Artificial Intelligence*, pages 326–337, Berlin, Heidelberg. Springer Berlin Heidelberg.
- Confavreux, B., Zenke, F., Agnes, E., Lillicrap, T., and Vogels, T. (2020). A meta-learning approach to (re)discover plasticity rules that carve a desired function into a neural network. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information*

- Processing Systems*, volume 33, pages 16398–16408. Curran Associates, Inc.
- Darlow, L., Regan, C., Risi, S., Seely, J., and Jones, L. (2025). Continuous thought machines.
- Duan, Y., Jia, Z., Li, Q., Zhong, Y., and Ma, K. (2023). Hebbian and gradient-based plasticity enables robust memory and rapid learning in rnns.
- Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016). RL²: Fast reinforcement learning via slow reinforcement learning.
- Egg, A. (2021). Online learning for recommendations at grubhub. In *Proceedings of the 15th ACM Conference on Recommender Systems*, RecSys '21, page 569–571, New York, NY, USA. Association for Computing Machinery.
- Elelimy, E., Szepesvari, D., White, M., and Bowling, M. (2025). Rethinking the foundations for continual reinforcement learning.
- Eryilmaz, S. B., Kuzum, D., Jeyasingh, R., Kim, S., BrightSky, M., Lam, C., and Wong, H.-S. P. (2014). Brain-like associative learning using a nanoscale non-volatile phase change synaptic device array. *Frontiers in Neuroscience*, Volume 8 - 2014.
- Ferigo, A., Cunegatti, E., and Iacca, G. (2024). Neuron-centric hebbian learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '24, page 87–95, New York, NY, USA. Association for Computing Machinery.
- Ferigo, A. and Iacca, G. (2023). Self-building neural networks. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, GECCO '23 Companion, page 643–646. ACM.
- Ferigo, A., Iacca, G., Medvet, E., and Pigozzi, F. (2023). Evolving hebbian learning rules in voxel-based soft robots. *IEEE Transactions on Cognitive and Developmental Systems*, 15(3):1536–1546.
- Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400.
- Gupta, A., Fan, L., Ganguli, S., and Fei-Fei, L. (2022). Metamorph: Learning universal controllers with transformers. In *International Conference on Learning Representations*.
- Gupta, A., Savarese, S., Ganguli, S., and Fei-Fei, L. (2021). Embodied intelligence via learning and evolution. *Nature Communications*, 12(1).
- Hebb, D. O. (1949). *The Organization of Behavior: A Neuropsychological Theory*. Wiley, New York.
- Heek, J., Levskaya, A., Oliver, A., Ritter, M., Rondepierre, B., Steiner, A., and van Zee, M. (2024). Flax: A neural network library and ecosystem for JAX.
- Hooker, S. (2021). The hardware lottery. *Commun. ACM*, 64(12):58–65.
- Huisman, M., van Rijn, J. N., and Plaat, A. (2021). A survey of deep meta-learning. *Artificial Intelli-*

- gence Review*, 54(6):4483–4541.
- Humplik, J., Galashov, A., Hasenclever, L., Ortega, P. A., Teh, Y. W., and Heess, N. (2019). Meta reinforcement learning as task inference.
- Javed, K. and Sutton, R. S. (2024). The big world hypothesis and its ramifications for artificial intelligence. In *Finding the Frame: An RLC Workshop for Examining Conceptual Frameworks*.
- Jax Docs (2024). Benchmarking jax code. <https://docs.jax.dev/en/latest/benchmarking.html>. Accessed: 2026-05-27.
- Jordan, J., Schmidt, M., Senn, W., and Petrovici, M. A. (2021). Evolving interpretable plasticity for spiking networks. *eLife*, 10:e66273.
- Kreuzberger, D., Kühl, N., and Hirschl, S. (2023). Machine learning operations (mlops): Overview, definition, and architecture. *IEEE Access*, 11:31866–31879.
- Krotov, D. and Hopfield, J. J. (2018). Unsupervised learning by competing hidden units. *CoRR*, abs/1806.10181.
- Lange, R. T. (2022). evosax: Jax-based evolution strategies. *arXiv preprint arXiv:2212.04180*.
- Leung, B., Haomachai, W., Pedersen, J. W., Risi, S., and Manoonpong, P. (2025). Bio-inspired plastic neural networks for zero-shot out-of-distribution generalization in complex animal-inspired robots.
- Matthews, M., Beukman, M., Lu, C., and Foerster, J. (2025). Kinetix: Investigating the training of general agents through open-ended physics-based control tasks. *The Thirteenth International Conference on Learning Representations*.
- Miconi, T. (2016). Learning to learn with backpropagation of hebbian plasticity.
- Miconi, T. (2023). Learning to acquire novel cognitive tasks with evolution, plasticity and meta-meta-learning.
- Miconi, T., Clune, J., and Stanley, K. O. (2018). Differentiable plasticity: training plastic neural networks with backpropagation. *CoRR*, abs/1804.02464.
- Miconi, T., Rawal, A., Clune, J., and Stanley, K. O. (2019). Backpropamine: training self-modifying neural networks with differentiable neuromodulated plasticity. In *International Conference on Learning Representations*.
- Miconi, T., Rawal, A., Clune, J., and Stanley, K. O. (2020). Backpropamine: training self-modifying neural networks with differentiable neuromodulated plasticity.
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.
- Najarro, E. and Risi, S. (2020). Meta-learning through hebbian plasticity in random networks. *CoRR*,

abs/2007.02686.

- Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63.
- Nikulin, A., Kurenkov, V., Zisman, I., Sinii, V., Agarkov, A., and Kolesnikov, S. (2023). XLand-minigrid: Scalable meta-reinforcement learning environments in JAX. In *Intrinsically-Motivated and Open-Ended Learning Workshop, NeurIPS2023*.
- Oja, E. (1982). Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273.
- OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. (2019). Solving rubik’s cube with a robot hand.
- Orchard, J. and Wang, L. (2016). The evolution of a generalized neural learning rule. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 4688–4694.
- Palm, R. B., Najarro, E., and Risi, S. (2020). Testing the genomic bottleneck hypothesis in hebbian meta-learning. *CoRR*, abs/2011.06811.
- Pedersen, J. W. and Risi, S. (2021). Evolving and merging hebbian learning rules: Increasing generalization by decreasing the number of rules. *CoRR*, abs/2104.07959.
- Rodriguez, H. G., Guo, Q., and Moraitis, T. (2022). Short-term plasticity neurons learning to learn and forget.
- Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, Volume 11 - 2017.
- Salimans, T., Ho, J., Chen, X., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.
- Snell, J., Swersky, K., and Zemel, R. S. (2017). Prototypical networks for few-shot learning.
- Soltoggio, A., Stanley, K. O., and Risi, S. (2017). Born to learn: the inspiration, progress, and future of evolved plastic artificial neural networks. *CoRR*, abs/1703.10371.
- Thangarasa, V., Miconi, T., and Taylor, G. W. (2020). Enabling continual learning with differentiable hebbian plasticity.
- Tian, Y., Zhao, X., and Huang, W. (2022). Meta-learning approaches for learning-to-learn in deep learning: A survey. *Neurocomputing*, 494:203–223.
- Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization

for transferring deep neural networks from simulation to the real world.

- Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.
- van Diggelen, F., Karagüzel, T. A., Rincon, A. G., Eiben, A. E., Floreano, D., and Ferrante, E. (2025). Emergent heterogeneous swarm control through hebbian learning.
- Vettoruzzo, A., Bouguelia, M.-R., Vanschoren, J., Rögnvaldsson, T., and Santosh, K. (2023). Advances and challenges in meta-learning: A technical review.
- Wang, F., Tian, H., Xiong, H., Wu, H., Fu, J., Cao, Y., Kang, Y., and Wang, H. (2022). Evolving decomposed plasticity rules for information-bottlenecked meta-learning. *Transactions on Machine Learning Research*.
- Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. M. (2016). Learning to reinforcement learn. *CoRR*, abs/1611.05763.
- Wang, L. and Orchard, J. (2019). Investigating the evolution of a neuroplasticity network for learning. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(10):2131–2143.
- Yaman, A., Iacca, G., Mocanu, D. C., Coler, M., Fletcher, G., and Pechenizkiy, M. (2021). Evolving plasticity for autonomous learning under changing environmental conditions. *Evolutionary Computation*, 29(3):391–414.
- Yaman, A., Iacca, G., Mocanu, D. C., Fletcher, G., and Pechenizkiy, M. (2019a). Learning with delayed synaptic plasticity. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19*, page 152–160. ACM.
- Yaman, A., Iacca, G., Mocanu, D. C., Fletcher, G., and Pechenizkiy, M. (2020a). Novelty producing synaptic plasticity. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, GECCO '20*, page 93–94, New York, NY, USA. Association for Computing Machinery.
- Yaman, A., Iacca, G., Mocanu, D. C., Fletcher, G., and Pechenizkiy, M. (2020b). Novelty producing synaptic plasticity. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, GECCO '20*, page 93–94. ACM.
- Yaman, A., Iacca, G., Mocanu, D. C., Fletcher, G. H. L., and Pechenizkiy, M. (2019b). Learning with delayed synaptic plasticity. *CoRR*, abs/1903.09393.
- Zargarbashi, F., Giuro, F. D., Cheng, J., Kang, D., Sukhija, B., and Coros, S. (2024). Metaloco: Universal quadrupedal locomotion with meta-reinforcement learning and motion imitation.