Neural Concept Learning

Combining Knowledge Using Modulating Masks

by Matthias Müller

Student ID: F234757

Loughborough University

22COP327: Artificial Intelligence Project

Supervisor: Dr Andrea Soltoggio

SUBMITTED ON 18th SEPTEMBER 2023

Abstract

Lifelong learning strives for developing agents that continuously learn during their lifetime, similar to biological learning. Supermasks have been shown to overcome catastrophic forgetting and even allow exploiting previous knowledge. However, the transfer has only been tested between similar tasks and is not interpretable or intuitive yet. This work shows that masks can be meaningful by resembling concepts, like a color or a shape. These concepts can further be combined in a way akin to human learning. A child can learn what green is, then learn what a ball is in an independent task without forgetting what green is. With minimal effort, the two concepts can be retrieved from a large selection of learned concepts and be combined to identify a green ball. A novel view on masks is presented that allows to mimic this learning principle to achieve forward transfer between tasks. Moreover, some intuition for the effectiveness of the approach is presented.

Contents

1 Introduction

The impressive advancements in deep reinforcement learning (RL) have led to algorithms capable of tackling complex tasks in specific environments. However, these algorithms often face limitations when it comes to sequential learning of multiple tasks or scaling up to handle increasingly intricate challenges. Lifelong Reinforcement Learning (LRL) aims to explore how agents can accumulate knowledge over time, leverage prior experiences to facilitate learning of new tasks, and ultimately build more adaptable AI systems.

1.1 Starting Situation

When trained on multiple tasks sequentially, traditional neural networks suffer from *catastrophic forgetting*. To achieve aforementioned goals, a lifelong learning agent must implement a technique to account for that like training a dedicated *Supermask* for each task. These have the additional property that they can be linearly combined. Wortsman et al. (2020) utilize this to find the corresponding mask to a task, even when the identity of the task is unknown. Ben-Iwhiwhu et al. (2022) combine masks during training to achieve forward transfer between tasks. However, the investigation of the previous knowledge exploitation capabilities is far from exhausted. Some curriculums used there contain quite similar tasks and it is not interpretable what has been combined, i.e. what pieces of knowledge are transferred between tasks. This thesis investigates if masks can be more meaningful, for example, if they can resemble concepts and further, if those concepts can be combined in a manner akin to human learning. For instance, let an agent be in a grid word confronted with the task to pick up a green item. In the next task, the agent shall pick up a blue item, then a ball, then a box. The agent must learn what all those concepts, i.e. different colors and shapes, mean in its world. However, if now the agent has to pick up the green ball, in theory, the agent should be able to solve the task very quickly, as it is already familiar with the necessary concepts *green* and *ball*. The main motivation of this thesis is to find out if, when, and how this can be achieved using masks.

1.2 Project Objectives

To summarize, in this thesis, I investigate the following questions:

- Can masks be trained such that they resemble meaningful knowledge or concepts?
- Can such masks be created such that they are combinable with each other?
- Is the algorithm able to detect useful masks for the current task among the already trained ones to combine them?
- Does combining those masks allow the agent to adapt quicker to new tasks, i.e. is there forward transfer?
- What can be inferred from the results with respect to how masks and their combination work?

1.3 Report Structure

To achieve this, first, an overview of continual learning techniques is presented in Section 2 to clarify the challenges and goals of learning tasks sequentially and to place the mask approach in the field. Section 3 presents how masks have been used in the literature and how they work. I compare different mask approaches to illustrate where masks come from and how their properties like the backbone network or the training algorithms have evolved. The section ends with explaining what has been done in terms of knowledge transfer in masks so far and, more importantly, what has not. Next, I design a custom environment optimized for investigating concept learning with masks in a RL-setting, see Section 4. Several experiments are conducted to answer the questions if masks can resemble concepts, in what scenarios they can or cannot be combined, if there is forward transfer and how it can be meassured, see Section 5. Finally, I discuss the results of the experiments and what apects of the algorithm should be further investigated. I present possible improvements to the algorithm, give some intuition what happens when masks are combined and why certain experiments did or did not work.

2 Background

This section introduces the foundation of the problem setting, namely continual learning or lifelong learning, its adaption to reinforcement learning and their categorizations.

2.1 Continual Learning

Continual Learning (CL) is the constant and incremental development of increasingly complex behaviours. This includes the process of building complicated behaviours on top of those already developed (Ring, 1997) while being able to reapply, adapt, and generalize previously learned abilities to new situations (Khetarpal et al., 2022). Take the example from the introduction. The agent could reapply the solutions to the task *pick up a green item* (written as `Pick up Green` from now on) and *pick up a ball* (`Pick up Ball`) to the more complicated task `Pick up Green Ball`. Generalizing in this example could mean extracting `Pick up Green` from the tasks `Pick up Green Ball` and `Pick up Green Box`. The main goal of CL is to allow a machine learning model to learn a series of diverse tasks sequentially without experiencing a significant decline in performance on tasks it has learned before. This phenomenon is widely known as catastrophic forgetting (French, 1999).

Various techniques address the challenges of CL, and they can be broadly classified into three approaches: replay based methods, regularization based methods, and parameter isolation methods (De Lange et al., 2021). While there may be other ways to classify these techniques, there is considerable overlap among them, such as in Wortsman et al. (2020). Replay methods involve the incorporation of samples from previous tasks, either real or artificially generated prototypes, during the learning of a new task. This integration of samples into the learning process helps alleviate forgetting, as demonstrated by the effectiveness of the iCaRL method (Rebuffi et al., 2017). In contrast, regularization methods impose constraints on specific parameters, preventing or penalizing changes that could adversely affect previously learned tasks, as seen in the work of Kirkpatrick et al. (2017). Lastly, parameter isolation methods allocate specific parts of the network to individual tasks, addressing forgetting in this manner. Masks (Zhou et al., 2019; Ramanujan et al., 2020) belong to the latter category. Wortsman et al. (2020) and Ben-Iwhiwhu et al. (2022) for instance train dedicated masks for different tasks on a fixed backbone network. Parameters for each task are thereby isolated and protected from forgetting. Approaches in this category have shown state-of-the-art results in classification, such as Progressive Networks (Rusu et al., 2016; Yoon et al., 2018), PackNet (Mallya and Lazebnik, 2018), Piggyback (Mallya et al., 2018) and Supermasks (Wortsman et al., 2020) The latter ones use masks and are further explained and compared in Section 3.2.

2.2 Lifelong Reinforcement Learning

Deep reinforcement learning has proven to be a powerful approach, showcasing its effectiveness in tasks like mastering video games (Mnih et al., 2013) and controlling robots (Lillicrap et al., 2015). However, similar to other neural networks, it faces the challenge of catastrophic forgetting in a lifelong learning setting without preventive measures. *Lifelong Reinforcement Learning* (LRL) incorporates concepts from continual learning into reinforcement learning. For instance, combining DQN (Mnih et al., 2013) with EWC (Kirkpatrick et al., 2017) led to the development of a lifelong learning DeepRL agent (Kessler et al., 2022). LRL shares aims with CL to achieve various objectives, including learning multiple tasks without forgetting, leveraging prior knowledge for efficient learning (forward transfer), improving performance on previous tasks while acquiring new knowledge (backward transfer), and efficiently utilizing model capacities. Due to the specific requirements of reinforcement learning, LRL has evolved into its own field (Ben-Iwhiwhu et al., 2022). Khetarpal et al. (2022) categorized LRL approaches into three groups. The first category focuses on explicit knowledge retention, similar to the CL approaches listed in Section 2.1, to stabilize learning. The second category, leveraging shared structure, decomposes problems into subproblems through function composition, as demonstrated by Griffiths et al. (2019). The third category, learning to learn, explores meta-learning techniques, see Section 2.3.

Masks in this project encompass various approaches, touching upon all of the mentioned strategies. For instance, masks can be employed to retain explicit knowledge about a single task, as demonstrated in Wortsman et al. (2020). While this approach prevents forgetting, it also restricts the potential for backward transfer, given that the masks remain unchanged after training on a task (Khetarpal et al., 2022). This dilemma is linked to the *Stability-Plasticity Trade-Off* (De Lange et al., 2021; Khetarpal et al., 2022), wherein the network must strike a balance between preserving parameters essential for solving previous tasks and allowing parameter adjustments to facilitate new learning. This work mainly investigates forward transfer capabilities of masks. Backward transfer is not considered here. Since adapting previously learned knowledge is a goal of CL (Khetarpal et al., 2022), see Section 2.1, this should be investigated in future work.

Approaches falling into the second category, which leverage shared structures, concentrate on training neural network modules that can be combined to handle a set of related tasks. By utilizing task-specific and specialized modules, these approaches enable efficient transfer learning across related tasks within the network (Khetarpal et al., 2022). An illustrative example is the work of Devin et al. (2017), where they demonstrate that breaking down neural network policies into task-specific and robot-specific modules facilitates information sharing between robots and tasks in a reinforcement learning context. This approach enables zero-shot generalization across various robots and tasks. However, note that devising modules that can be effectively combined in more complex scenarios is far from being straightforward. As highlighted by Khetarpal et al. (2022), a "chicken and egg" challenge arises. Even when the environment remains stationary, the process of learning modules and determining how to combine them becomes a non-stationary problem, as the modules themselves and their combinations evolve during the learning process. This issue is closely connected to the third category of approaches, known as learning, how to learn or meta-learning. It constitutes a distinct and extensive field in its own, encompassing more than just lifelong reinforcement learning techniques, and will be elaborated upon in a dedicated section.

2.3 Meta-Learning

Within the realm of deep RL literature, a meta-learning framework has been introduced to divide the recursive learning process into two distinct phases: the *inner loop* for rapid time-scale learning and the *outer loop* for the slower process of learning about learning (Bengio et al., 2013a). Hospedales et al.

(2021) propose a taxonomy that explores various aspects of meta-learning. One aspect, termed *Meta-Representation*, focuses on identifying *what* constitutes the meta-knowledge, encompassing information about the inner model learned and optimized by the outer loop. This meta-knowledge can take different forms, such as initial parameters (Finn et al., 2017), the loss function (Houthooft et al., 2018), hyper-parameters like regularization strength (Franceschi et al., 2018), and more (Hospedales et al., 2021). Another example of meta-knowledge involves embeddings, where the raw input is processed by an outer model to generate embeddings, upon which the inner model operates. For instance, in a classification task, Snell et al. (2017) employ a neural network to create embeddings used to form prototypes for each class. In zero-shot and few-shot scenarios, samples are classified based on nearest neighbors in the latent space. A similar, yet more complex, approach could be employed with the proposed mask approach in the long run. Here, the task is fed into an outer loop, such as in natural language processing, resulting in an embedding that dictates which masks from the available collection are selected to be combined in the inner model.

The second axis in Hospedales et al. (2021), termed *Meta-Optimizer*, delves into *how* the meta-knowledge is acquired, i.e., the inner workings of the outer loop. For differentiable problems, gradient-descent could be employed (Finn et al., 2017), whereas non-differentiable problems might benefit from RL techniques or evolutionary search.

Finally, the *Meta-Objective*, as the third axis, establishes the *why* behind using the outer loop, determining the goal of the meta-learning process. Examples include few-shot learning (Finn et al., 2017), robustness to domain shift (Li et al., 2019), or adversarial defense by generating an adversarial validation set (Goldblum et al., 2020).

2.4 Transfer Learning

In transfer learning, a concept closely related to meta-learning, past experience gained from a source task is leveraged to enhance learning performance on a target task, benefiting speed, data efficiency, or accuracy. The key distinction from meta-learning lies in the method of obtaining the prior knowledge. In transfer learning, the prior knowledge is extracted through vanilla learning on the source task without involving a meta-objective (Hospedales et al., 2021). Consequently, the modulating masks approach presented in Ben-Iwhiwhu et al. (2022) can be categorized as transfer learning rather than meta-learning, as it does not use an outer loop optimizing meta-knowledge. This also holds for this work, given its foundation on the work of Ben-Iwhiwhu et al. (2022).

3 Masks in Machine Learning

Masks emerged from two directions in the literature. One was concerned with pruning and investigating what random networks can achieve, see Section 3.1. The other one came from the continual learning perspective and looked for ways to fit multiple tasks in a single network, see Section 3.2 and 3.3. In this section, I present approaches from both directions and compare them at a level beyond what I have seen so far to illustrate how properties like the backbone network or the mask finding algorithms evolved. At the end, the current limitations are described in detail.

3.1 Finding Sparse Subnetworks

Neural networks tend to be over-parameterized and, thereby, allow compression both on layer (Denil et al., 2013) and network level (Li et al., 2018). Based on that Frankle and Carbin (2018) proposed the

Lottery Ticket Hypothesis: *A randomly-initialized, dense neural network contains a subnetwork that is initialized such that — when trained in isolation — it can match the test accuracy of the original network after training for at most the same number of iterations.*

They present a simple algorithm to find those sparse subnetworks in larger networks. Their research reveals that highly pruned networks, where 95% or more of the weights are pruned and set to 0, exhibit no decline in performance compared to their unpruned counterparts. Interestingly, networks subject to moderate pruning, involving 50% to 90% weight reduction, often perform even better than the original dense network. This trend becomes more pronounced with deeper networks. They achieved this by first training a randomly initialized network and then pruning parameters of smaller magnitude. The idea is that connections with a small absolute weight do not contribute much to the result of the recipient node anyway. Afterwards, the network is retrained with the original initialization while the pruned parameters are frozen at zero. Pruning the network with weights w is implemented by determining a binary mask $m \in \{0,1\}^{|w|}$ that has the same shape and number of parameters as the weights of the network. The weights w and the binary mask m are multiplied elementwise ($\odot$) when calculating the output of the network: $f(x; w \odot m)$.

Zhou et al. (2019) discovered that winning tickets of the *Lottery Ticket Hypothesis* achieve better than random performance even when the weights selected by the mask are set to their initial values and are not retrained. Masks that can be applied to a randomly initialized, untrained network, are called *Supermasks*. To find them, they still train the weights of the neural network. Next, the masking criterion determines the connections in the network that will be taken by the mask, for example, taking the connections whose weights got transformed into large absolute values during training. Besides discovering that the mask performs well even when applying it to the original untrained network, they found out that the remaining weights can even be set to a constant rather than their initial values as long as the sign stays the same. Furthermore, Zhou et al. (2019) show a probabilistic method to learn those masks without learning any weights at all. Each weight has assigned a trainable parameter p which is the probability that the corresponding weight is taken in a forward pass. The weights remain fixed.

Ramanujan et al. (2020) argue that even if the parameters for the probabilities are fixed, the algorithm will likely never observe the same subnetwork twice. As such, the gradient estimate becomes more unstable, which could complicate the training. In contrast, they propose the *Edge-Popup Algorithm*. Each weight has an associated score value. In a forward pass, only the weights with the top $k\%$ scores are taken. To avoid zero gradients and allow a deactivated weight to popup again, backpropagation is combined with the straight-through gradient estimator (Bengio et al., 2013b; Courbariaux et al., 2015), i.e. the identity function is used for the gradient of the binary mask even if a weight is not taken. For k, values between 30% and 70% produce the best results. Ramanujan et al. (2020) make a combinatorial argument for this. The number of possible subnetworks for a given k can be calculated by $\binom{n}{kn}$, where n is the number of weights. This number is maximal when k is around 50%. For initializing the weights, they found that the signed kaiming constant with a random sign is superior. For each weight, they sample from the set $\{-\sigma_k, \sigma_k\}$ where σ_k is the standard deviation for the kaiming normal distribution (He et al., 2015).

3.2 Masks for Continual Learning

The main effort in the works of the previous section was finding smaller sub-networks with a competitive accuracy to solve a specific task. When applying this to a multi-task setting, it can be seen as a parameter isolation technique, one class of *Continual Learning* methods, see Section 2.1. Examples for this are PackNet (Mallya and Lazebnik, 2018), Piggyback (Mallya et al., 2018) and Supermasks in Superposition (Wortsman et al., 2020).

In PackNet (Mallya and Lazebnik, 2018), the process begins with training a neural network for a specific task. Subsequently, an iterative pruning procedure is applied, involving the removal of weights with the smallest magnitudes. This procedure shares similarities with the methodology shown by Frankle and Carbin (2018), without resetting the remaining weights to their initial values when pruning. After completing training of a task, a mask is generated based on the surviving weights, which are then frozen. This strategy helps mitigate *Catastrophic Forgetting*, as the network capable of solving the original task can be reconstructed using the established mask at any time. Note that, much like other masking approaches, this technique imposes constraints on the potential for backward transfer, as the weights are frozen. Since the selected weights can be reused by future tasks, without changing them, forward transfer may occur.

Kang et al. (2022) point out that since the weights with largest magnitude are selected and then frozen, this may lead to an increased bias for each future task. Even if a particular weight is not helpful or even counterproductive for a future task, it is likely to be selected again because it has a high magnitude. Kang et al. (2022) circumvent this problem. Weights are still trained and fixed once selected by a mask. However, instead of using the absolute value of the weights as the masking criterion, they use score values in a similar fashion as the edge-popup algorithm (Ramanujan et al., 2020).

In Piggyback, Mallya et al. (2018) account for this problem in a different fashion. Instead of training weights, a fixed backbone network is used similar to Zhou et al. (2019) and Ramanujan et al. (2020). However, instead of a randomly initialized network, a pre-trained model is used such as VGG-16 (Simonyan and Zisserman, 2014) or ResNet-50 (He et al., 2016) trained on ImageNet. To determine the mask of a task, each weight parameter in the network is assigned a trainable real value. This is essentially the same as score values in the edge-popup algorithm (Ramanujan et al., 2020). In contrast, while Ramanujan et al. (2020) select the top $k\%$ of score values, Mallya et al. (2018) use a threshold function, i.e. a weight is taken if its corresponding score succeeds a certain value.

Wortsman et al. (2020) combine ideas from several of the already mentioned papers and extend the use and handling of masks. They employ the fixed signed Kaiming constant backbone network and the edge-popup algorithm from Ramanujan et al. (2020). So far, all previous continual learning masking approaches require the identity of the task, i.e. the model is told what the current task is. Wortsman et al. (2020) introduce a method how the model can infer the current task. Let's consider the case that masks have been trained for multiple tasks. The model is confronted with one of these tasks but does not know which one. When presented with data from the current task, a for this task trained mask is expected to have a confident output, i.e. a low entropy distribution. For instance, if a mask is trained on recognizing digits, the output is expected to be quite high for one label and close to zero for the other digit labels. If the same mask is presented a picture of a dog in a cat-dog-classification task, the same mask will produce random values in the output vector, as the mask has never seen such an input distribution before. Hence, the inference of the task-identity can be seen as an optimization problem of minimizing entropy.

So far, masks were only used as a parameter selection and isolation mechanism. Wortsman et al. (2020) utilize a novel property of masks, their linear combination. Let k be the number of tasks learned so far. Each mask m_i is multiplied by a coefficient α_i, where $\sum_{i=1}^{k} \alpha_i = 1$. The masks are superimposed forming a new mask, which is not binary any more, but consists of values between zero and one. Wortsman et al. (2020) call this *Superposition*. The goal is to find α coefficients that minimize the output entropy, while the weights and the individual masks stay fixed. At the beginning, all masks are equally likely to be the correct one, so initially all $\alpha_i = \frac{1}{k}$. The mask with the highest coefficient is selected for the task. This can be determined using gradient decent, either one-shot, where only one gradient decent step is executed, or with a binary search, where half of the masks are eliminated after each step. Furthermore, Wortsman et al. (2020) show that the method can be improved by adding *Superfluous Neurons* to the last layer. These are output neurons that do not belong to any class. Due to the standard cross-entropy loss, the values of these neurons are pushed to zero. Note that omitting all incoming connections to those neurons by not including them in the mask is not a trivial solution here. Since the softmax function is applied in the end, the values of the superfluous neurons are encouraged to be large negative values, such that they are close to zero after softmax. As each mask is optimized to achieve this given the input distribution of data they are trained on, this can be used to improve the inference of task identity.

3.3 Masks for Lifelong Reinforcement Learning

In contrast to all papers shown above, which investigate masks on supervised classification problems, Ben-Iwhiwhu et al. (2022) apply masking approaches to (lifelong) reinforcement learning (LRL), called *Modulating Masks*. Furthermore, they extend the idea of linearly combining masks (Wortsman et al., 2020) to achieve forward transfer, one goal of LRL, see Section 2.2. Their assumption is that prior knowledge is stored in already trained masks. To exploit this when being faced with a new task, a linear combination of trained masks, which are fixed, and a random trainable mask are used to initialize the new mask. As in the work of Wortsman et al. (2020), all masks, the already trained ones as well as the new random mask, get assigned a coefficient β_i and are added up. Probably, β was chosen to indicate the similarity to α as a coefficient in a linear combination in Wortsman et al. (2020), while stressing that it is used differently. Regardless of the reason, since this work is mainly based on their work, β will be used from now on in this work as well. A softmax function ensures that the β-values add up to 1.

They suggest two methods to initialize the weights. First, all trained masks and the random mask have initially the same weight, e.g. when training the third task, the two trained masks from the task before and the random mask are each assigned a weight of $\beta_i = \frac{1}{3}$. They call this *Mask Linear Combination.* Second, if the number of known tasks increases, the weight of each mask decreases. To avoid overweighting the trained masks, the weight of the random mask is always set to 0.5, the remaining 0.5 are divided among the trained masks. They call this *Balanced Linear Combination.* Only the β-values of the masks as well as the score parameters of the random masks are updated during training. After training, the resulting mask of the linear combination is calculated and stored as an individual mask, which can be used for combinations in future tasks. Section 3.4 provides an example for the algorithm. As masking method, they use a threshold function for the score values, similar to Mallya et al. (2018). Note that using previous masks in the initialization process induces a bias. However, here it can be overcome easier as in PackNet (Mallya and Lazebnik, 2018), see Section 3.2. In PackNet, high magnitude weights get selected and are frozen, meaning that they are more likely to get selected in future tasks as well. If a weight is counterproductive, it must be balanced out somehow or exceeded by sufficiently many other weights. In the modulating masks approach (Ben-Iwhiwhu et al., 2022), if a weight from previous tasks is not helpful, it can be removed by two ways. First, the score value of the corresponding weight can be reduced in the random mask, as it gets added in the linear combination before applying the masking threshold. Second, the coefficient of the corresponding mask could be reduced. A mask has a separate β-value for each layer. Therefore, in this case, the whole layer of this mask would be affected.

Ben-Iwhiwhu et al. (2022) showed that while binary masks can be applied to discrete RL problems, they struggle in RL environments with continuous action space. To approach this problem, they suggest continuous modulating masks by changing the masking criterion. If a score value exceeds the threshold, the score value itself is taken as the mask value, rather than one.

3.4 Linear Combination

Figure 1 illustrates how the linear combination of mask works. It shows two time axes. Training progresses from top to bottom. Calculating the current mask while training this task can be read from left to right. The explanation starts in the top left corner. Before training of task one begins, a new random mask is being created. The color gray represents initialization. The mask consists of score values. The are positive and negative real values. A thicker line represents a higher value, a thinner line a lower value. The score values are put through a threshold function to create a binary masks which is elementwise multiplied with the backbone network to get the final subnetwork for the current task. Figure 1 visualizes that only the weights of the backbone network are used in the final masks whose corresponding score value in the mask is above the threshold. All lines of the backbone have the same

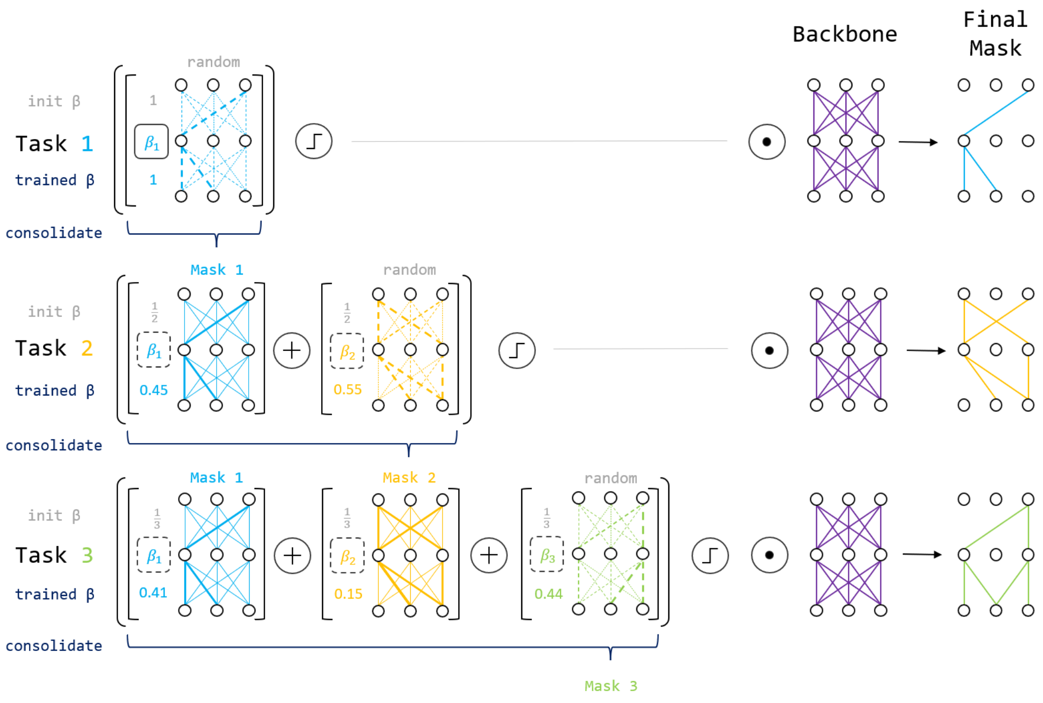


Figure 1: Visualization of the linear combination mask algorithm. The thickness of a line indicates how high the value is. The color gray represents initialization. Trainable parameters are shown as dashed lines, fixed parameters as solid lines. The presentation is inspired by Ben-Iwhiwhu et al. (2022). Readability was increased, the new layout represents the functionality better and important properties such as trainability of parameters were newly included.

thickness, as all weights are initialized with the same constant. Note that in practice the constant is assigned a random sign, which is neglected here. When training of the first task has finished, the mask is consolidated to the solution for the first task. When training the second task, the consolidated mask for task one is combined with a new random mask. The β-coefficients are initialized with $\frac{1}{2}$ each. Solid lines indicate that the parameter is fixed, whereas trainable parameters are dashed. Note that only the β-coefficients and the score value of the new random mask are trainable, while previous masks, here `Mask 1`, and the backbone network are fixed. The graphic shows how the combination works. Note that in the final mask the weight connecting the center left neuron with the top right one is kept even though the corresponding score value in the orange masks is not particularly high, as it is not small enough to outweigh the corresponding score value in the blue mask. When training for the second task has finished, all masks are combined a last time with the trained β-coefficients to be consolidated to `Mask 2`. The solution mask for task two can now be generated any time by applying the threshold function to `Mask 2` and multiplying it to the backbone network. The same process is repeated over and over for each new task. Figure 1 depicts the *Mask Linear Combination* approach, as the β-values are initialized to $\frac{1}{3}$ when training task three, see Section 3.3. In this example, β_2 was decreased to 0.15. Therefore, `Mask 2` barely contributes to the final mask, which is now mainly a combination from `Mask 1` and the new random mask.

3.5 Creating Masks vs. Learning Own Networks

A question that could arise is why should we try to find masks if we could just learn separate networks for each task. In this section, I want to give some intuition why creating masks is not the same as learning own networks. The first reason is storage requirements. In a regular network, a real value must be stored for each parameter. In masking approaches, only one network must be stored. For a binary mask, only one bit is necessary for each parameter. Furthermore, Wortsman et al. (2020) store the binary masks in a *Hopfield Network* to achieve sub-linear memory requirements with respect to the number of tasks. If a fixed backbone network is used, it is sufficient to store its seed, rather than the whole network (Zhou et al., 2019), to further reduce memory requirements. Kang et al. (2022) compress their binary masks with Huffman coding. Even if the masks are continuous, as introduced by Ben-Iwhiwhu et al. (2022), they can still be sparse meaning that a real value must only be stored for the weights taken in the mask. The second motivation for using masks is that they can be combined, for instance, as a linear combination (Wortsman et al., 2020; Ben-Iwhiwhu et al., 2022). This property allows masks to be retrieved even when the task identity is unknown without trying all masks individually (Wortsman et al., 2020), see Section 3.2 and can achieve forward transfer (Ben-Iwhiwhu et al., 2022). A comparable approach with training networks are *Progressive Networks* (Rusu et al., 2016).

Approach	Fixed Backbone Network	Masking Criterion	CL	Use Case	Linear Combination
Lottery Ticket Hypothesis (Frankle and Carbin, 2018)	no	weight magnitude	no	classification	no
Deconstructing Lottery Tickets (Zhou et al., 2019)	signed constant	probabilistic	no	classification	no
What's Hidden in Random NN (Ramanujan et al., 2020)	signed constant	edge-popup: top k% scores	no	classification	no
PackNet (Mallya and Lazebnik, 2018)	no	weight magnitude	yes	classification	no
Forget-Free CL (Kang et al., 2022)	no	top c% scores	yes	classification	no
Piggyback (Mallya et al., 2018)	pretrained	score threshold	yes	classification	no
Supermasks in Superpositions (Wortsman et al., 2020)	signed constant	edge-popup: top k% scores	yes	classification	task inference
Modulating Masks (Ben-Iwhiwhu et al., 2022)	signed constant	score threshold	yes	RL	forward transfer

Table 1: Comparison of mask approaches in literature

3.6 Overview of Masking Approaches

To get an overview how the different mask approaches differ, they are listed and compared in Table 1. The criteria are:

- is the masked network a fixed backbone or are the weights trainable as well,
- how are the masks created, i.e. what condition must be met for a weight to be in the mask,
- does the paper investigate single tasks or a continual learning setting,
- what is the type of the investigated problems, i.e. a supervised classification scenario or a reinforcement learning agent,
- are masks linearly combined, i.e. put in *Superposition*.

Other aspects to compare the approaches could be how masks are stored, for example in *Hopfield Networks* (Wortsman et al., 2020), and how gradients are propagated, e.g. the straight-through gradient estimator (Bengio et al., 2013b; Courbariaux et al., 2015). Those are omitted here, as they are less important and distinctive.

3.7 Forward Transfer Using Modulating Masks

In their paper Ben-Iwhiwhu et al. (2022) show that previous knowledge can be exploited using their linear combination of masks. Two environments they use among others are CT-Graph (Soltoggio et al., 2023) and Minigrid (Chevalier-Boisvert et al., 2023). To understand what they are investigating, the two environments and the corresponding curriculums are explained.

The CT-Graph, which stands for configurable tree graph, (Soltoggio et al., 2023) is a sparse reward, discrete action space environment with configurable parameters that define the search space. The environment is represented as a graph, where each node is a state represented as a 12×12 gray scale image. There are different types of nodes that require the agent to perform different actions. Each graph contains one start and end node with a single trajectory connecting the two. Reaching the goal results in a reward of 1. In all other cases, there is no reward. The deeper the graph is, i.e. the longer the trajectory is, the lower the probability of receiving a reward. To solve the task consistently, the agent must remember how to act in each state which can be distinguished by their corresponding gray scale image. They create a curriculum of increasingly difficult tasks by starting with a small configuration and extending the graph further. In each following task, the trajectory to the last goal state forms a partial solution to the new goal state. Thereby, they analyse how the linear combination of masks helps to exploit the fact that previous masks have already learned a partial solution.

Minigrid (Chevalier-Boisvert et al., 2023) is also a sparse reward, discrete action navigation environment setup as a partially observable grid world. Besides predefined tasks with varying complexity, it offers a framework to design own worlds and tasks, which will be extensively used when designing the custom environments for this work, see Section ??? In the work of Ben-Iwhiwhu et al. (2022), the task is always to navigate to a goal location. There are barriers consisting of walls, which cannot be entered by the agent, or lava, which kills the agent. In each task the goal location as well as the barriers stay at the same position, so the agent is mainly incentivised to learn the trajectory to the goal. The last four of the ten tasks in their curriculum have the same configuration of the barriers, i.e. their location, as the first four, but use lava instead of walls.

They could show that the different masking methods they proposed, i.e. *Mask Linear Combination* (Mask_{LC}) and *Balanced Linear Combination* (Mask_{BLC}), see Section 3.3, learned the tasks quicker than the elastic weight consolidation (EWC) algorithm (Kirkpatrick et al., 2017), see Section 2.2. Also since task labels were used, i.e. the task identity is known, the mask approach has a perfect recall of the performance of each already learned task. However, the investigation of the previous knowledge exploitation capabilities in this paper is far from exhausted. Besides their linear combination of mask approaches, they also implemented a method called *Mask Random Initialization* (Mask_{RI}), which employs only the new mask with random score values without superimposing any previous masks. This builds a useful baseline to determine how much of the performance can be contributed to the superposition of masks and what can be attributed to the masking algorithm itself. For instance, in the CT-Graph, previous knowledge was exploited because both methods Mask_{LC} and Mask_{BLC} adapt quicker to the new task than Mask_{RI} (can be seen in Figure 3 right, 4 right as well as Table 2 and 3 of their paper). However, the beta values were hardly used in the learning process (Figure 16 in their paper). One intuition could be that since a task is a superset of its predecessors, later tasks have a higher weight as they might be more useful for solving the current task. However, apparently, all learning is done in the random mask, as the beta coefficients hardly differ from their initialization, and the previous masks are all used as priors for the new mask. One explanation could be that no previous mask has a negative impact. Therefore, the agent must not select what masks are actually useful, but this is speculation so far.

In their Minigrid curriculum, it is hard to tell what the agent has actually learned and what knowledge is combined. For example, the task is always to get to the goal location. Therefore, one piece of knowledge could be that if the goal is in sight, try to navigate towards it. If there is a barrier in between, this might mean that the agent has to turn or move such that the goal cannot be seen any more in the next step. Note that the agent has no memory. Therefore, it must learn how to act in certain configurations of barriers, i.e. learn the trajectory. This does not necessarily mean that the agent generalizes for unseen tasks. Furthermore, since the last four tasks of the curriculum have the same configuration of barriers just with lava instead of walls, this may test if the agent learned a trajectory. However, besides walls being not enterable and lava killing the agent, they differ in another aspect in the Minigrid framework. Walls block the view of the agent while lava does not. Therefore, the states the agent is in differs more. One indicator that the agent actually learns a trajectory is that for solving task 7, which is the first lava task, the first mask, which corresponds to the task with the same configuration of barriers just with walls, is utilized above average, see Figure 16 in their paper. The same holds for task 8 and mask 2. The effect cannot be observed again, even though the same relationship holds for 9 and 3 as well as 10 and 4. A human would probably learn differently. The knowledge that stepping into lava is bad can be considered one atomic piece of knowledge. One could expect that when learning a lava task, e.g. T10 in their curriculum, the agent detects that masks of other lava tasks (T7 - T9) are useful. This cannot be observed in Figure 16 in their paper. Therefore, what lava is or means, was probably not learned by the agent or at least not learned in a reusable fashion.

Moreover, the composition of tasks can be questioned. Since a mask is trained for each task, and masks, or to be more precise, their layers, are the atomic elements that are combined, the definition, what is considered a task, is quite relevant in this setting. For example, one could argue that the used Minigrid curriculum only consists of two tasks: Find the goal with wall barriers and find the goal in lava setting.

To summarize, in their CT-Graph experiments, they showed that masks can be used to learn a task which is broken down into increasingly complex subtasks. However, the most useful masks, i.e. the ones of the most complex subtasks are not used above average. In the Minigrid setup, the agent probably just memorized certain parts of the solution rather than learning concepts like what is lava. These findings motivate this work to further investigate the potential of linearly combining masks to achieve forward transfer. The objectives for this project, as already stated in Section 1.2 originate from the reasoning described in this section. Therefore, before beginning to discuss the novel contributions of this work, let's recapitulate the objectives:

- Can masks be trained such that they resemble meaningful knowledge or concepts?
- Can such masks be created such that they are combinable with each other?
- Is the algorithm able to detect useful masks for the current task among the already trained ones to combine them?
- Does combining those masks allow the agent to adapt quicker to new tasks, i.e. is there forward transfer?
- What can be inferred from the results with respect to how masks and their combination work?

4 Environment

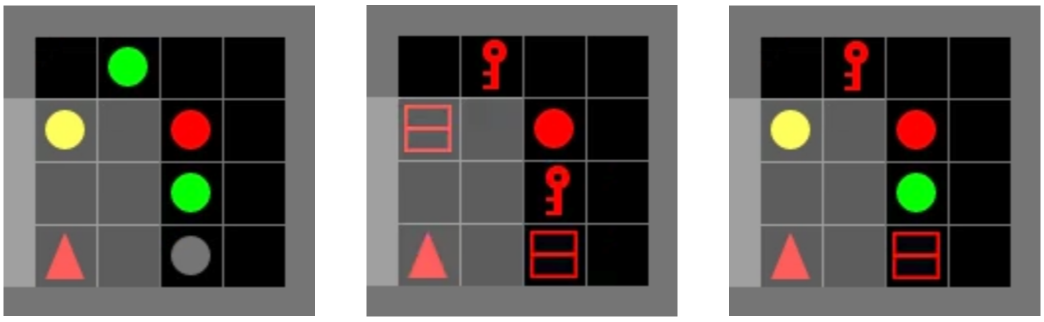
In order to investigate if masks can resemble meaningful and combinable concepts, the setup of tasks is important, as each task will result in a mask. Therefore, we want a task that can be solved by combining the solutions from two previous tasks. Ideally, the pieces of knowledge, which have been learned in previous tasks and shall be combined, should be disjunct, i.e. one shall not be partially learned by another. If this does not hold for the concepts that shall be combined, it is difficult to assess the contribution of the combination. Besides walls, lava and goals, other items exist in the Minigrid framework as well. Each item has one of six colors and is one of three types: a ball, a box or a key. Furthermore, items can be picked up by the agent. Color and shape provide the desired property. They can be learned independently from each other. In this section, the development process of the used environment is described. Thereby, the properties are introduced in an incremental fashion to understand why the final environment is designed how it is.

4.1 Combinable Tasks

To learn color and shape, these properties must be part of the task description in some way. Navigating to the goal tile as done in Ben-Iwhiwhu et al. (2022) is not an option, as the goal is a separate type in the Minigrid framework and only items have a color and shape, i.e. balls, boxes and keys. Therefore, the agent must interact with them in some way. When exploring the framework, there are several options:

- Go to a specific item, i.e. the agent stands right in front of the desired item.
- Pick up a specific item.
- Pick up a specific item and drop it next to another specific item.

The first one has the disadvantage that it could be solved solely with a good exploration technique. The reason is that the agent can try standing in front of each item without having to make a final decision. This can be circumvented by employing the second option. The agent has to pick up the item that the task asks for. If the picked up item is correct, the agent receives



(a) Distractors are balls but differ in color.

(b) Distractors are red but of a different shape.

(c) Distractors differ either in color or shape.

Figure 2: Visualization of a potential curriculum. In all tasks, the goal is to pick up the red ball. The distractor objects differ. The red triangle represents the agent looking in a direction. This setup has disadvantageous properties and is not used in the experiments.

a reward, otherwise the agent gets killed, i.e. the run is terminated, without any reward. Thereby, an exploration solution is prevented. The third option is a more difficult version of the second one. Since this work investigates the principle of combining tasks on a fundamental level, the problem shall be as relaxed as possible, see Section 4.4.

Next, we have to decide what item(s) the agent has to pick up. The first option is that the agent learns color and shape of the same item in different tasks. For example, the agent shall always pick up the red ball, see Figure 2. This can be achieved with a curriculum consisting of three levels, all of which contain exactly one red ball and several distracting objects. In the first level, all distracting items are other balls, see Figure 2a. Therefore, the only distinct property of the items is their color. Thereby, the agent must learn what *red* is. In the second level, all distracting items have a different shape, i.e. are either a box or a key, but all of them are red, see Figure 2b. To solve this level, the agent must learn to distinguish the shape *ball* from the others. The masks trained on those two tasks could be combined to solve the third level. It contains both kinds of distracting items, other colored balls and red boxes and red keys, see Figure 2c. While the last task is a combination of its two predecessors, the curriculum has a disadvantage. The desired properties are not properly separated, as the goal is always the same, i.e. always pick up the red ball. For example, in the first task, the agent must distinguish red from other colors. However, the agent might also learn to pick up balls in general, as picking up an empty tile does not give a reward. Thereby, the agent would not learn to pick up red items as intended but the red ball in specific. This could allow the agent to solve the third task directly without requiring knowledge from the second task. Another possibility is that, in the first level, the agent might learn that all colors except red are bad and shall never be picked up. If the corresponding mask is used to create the mask for the second task, we cannot know if this knowledge is also transmitted. Since the second level does not contain any colors except red, there is neither an incentive for the agent to keep this information for the new mask nor to avoid it. The knowledge of avoiding colors other than red could persist in the second mask along with the information to pick up a ball. This would also result in the correct solution for the last task without relying on the combination.

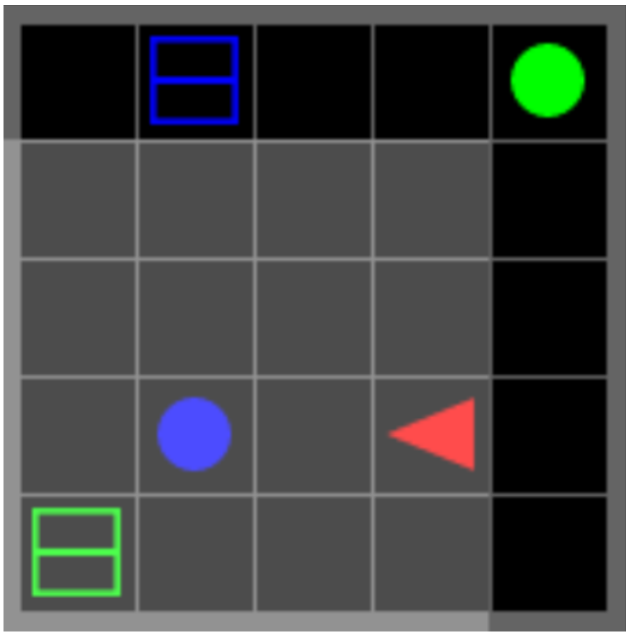


Figure 3: Example visualization of the environment used in the experiments. The red triangle represents the agent looking in a direction. Its first person field of view is highlighted by the light gray shade. The environment always contains these four items, a green ball and box and a blue ball and box.

To solve these problems, the goal cannot be the same in each task and the environment in each task ideally contains the same items[1]. Therefore, there must be items of different types and colors. To relax the problem as far as possible while still allowing nice combination possibilities, we need two colors and two types. For this work, the environment for each task contains: a green ball, a green box, a blue ball and a blue box. This setup allows a task suite with nice properties:

1. Pick up any item!
2. Pick up a green item!
3. Pick up a blue item!
4. Pick up a ball!
5. Pick up a box!
6. Pick up the green ball!
7. Pick up the blue ball!
8. Pick up the green box!
9. Pick up the blue box!

The first task, the *Root Task*, just learns what an item is and that picking them up results in a reward. All four items are valid. The tasks 2 - 5 are called *Base Tasks.* They ask for a specific property, either a color or a shape. For each task, two of the four items are valid. The remaining tasks are *Complex Tasks*, as they aim for a specific item. Each of those tasks only accepts one valid item and can be solved by combining two base tasks.

Figure 3 shows how the final environment looks like. The agent is represented by the red triangle pointing in the current direction of the agent. Each level, regardless of the specific task, contains the same four items. The agent experiences the environment from a first person perspective, a square field of view of up to 5×5 if not blocked by walls, illustrated by the light gray

[1]Note that this only holds since the tasks shall be to pick up a specific item. In other scenarios this might not be the case.

shade. This has an advantage and a disadvantage. The agent does not need to learn to locate itself on the input as the same input neurons will always represent the same tile from its own perspective. The downside is that the agent can only see items in its field of view. For instance, in Figure 3, it can currently only see the blue ball and the green box.

This setup solves the problems of the other environment aiming for always picking up the red ball, see Figure 2. For instance, if the agent is confronted with the task `Pick up Green` and it focuses only on the green ball, it would come up with the solution for the complex task `Pick up Green Ball`. However, the agent has no incentive to do that. Neglecting the green box for `Pick up Green` would reduce the average reward, as it will be closer to the agent than the green ball in some seeds. The second problem was that knowledge of a previous task can life in a new mask if it has no influence on the current task. This is hardly possible here as any knowledge will influence the performance of the current task. For example, say the agent has learned `Pick up Green` and now trains on `Pick up Ball`. Since there are always the green box as well in the environment, taking the knowledge to go for green items also has a negative influence. Therefore, the agent has an incentive to reduce the β-value of the mask for `Pick up Green`, thereby, correctly identifying that the mask is not useful for the current task. Therefore, knowledge from previous tasks cannot life in new masks unintended as in the environment discussed earlier. Either knowledge from a previous task is useful, then it is taken, or it is obstructive, then it is avoided. Through the specific design of the environment, there is no neutral knowledge at least in the first layer. This is its main advantage and allows proper analysis of combining masks. Carefully designing the environment having those properties and thinking of possible consequences is the first major contribution of this project.

4.2 Task Embeddings

The tasks can be encoded in a nice fashion, aligning them in a two dimensional space. The first axis resembles the color property, let 1 be green and -1 be blue. Shape is resembled by the second axis, 1 for ball and -1 for box. If a property is not important in a task, 0 is used. For example, the task `Pick up Green` can be encoded as $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$, and `Pick up Green Box` as $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$, see Figure 4. These encodings could be used in training. For example, they could be fed as input to the agent about the current task. Thereby, the agent does not need to understand natural language, thereby, relaxing the problem. The embeddings could also be used for the outer loop in a meta-learner, see Section 2.3. In this work, these embeddings are not used. The agent does not know the current task. Instead, it has to figure it out using reinforcement learning.

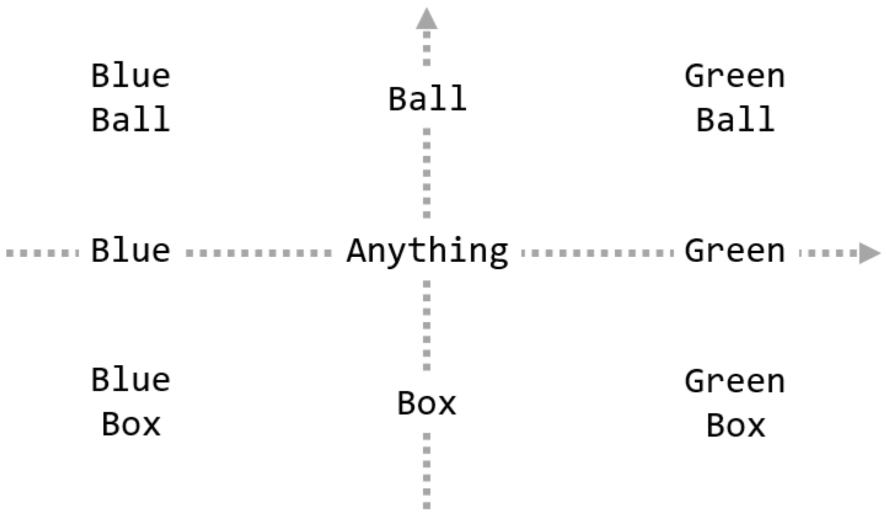


Figure 4: Visualization of possible embeddings of the tasks in a two dimensional space. The x-axis encodes the color, the y-axis the shape of the targeted item. The value 0 on any axis means any color/shape is allowed.

4.3 Controlled Randomness

To ensure that the agent actually learns color and shape, the positions of the items and of the agent are reshuffled after each run, i.e. when the agent picks up any object regardless if it is valid or not, or when the maximum number of steps per run is exceeded. Thereby, the agent does not implicitly learn a trajectory to the target item. While it might be possible to craft levels such that combining trajectories leads to success, this work is interested in learning concepts like *green* or *blue*. Note that avoiding trajectories is an important difference to the work of Ben-Iwhiwhu et al. (2022), see Section

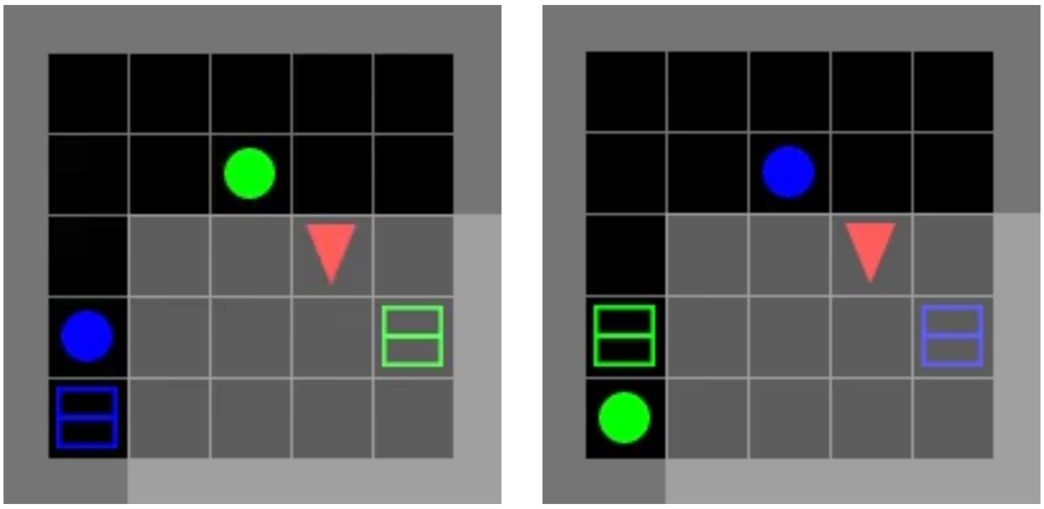

(a) Task: `Pick up Green` (b) Task: `Pick up Blue`

Figure 5: Both levels are created with the same seed. In both cases the valid items spawn at the same positions, in this seed next to agent. This increases comparability between tasks. Note that while the corresponding distracting items are also at the same positions, i.e. in the bottom left corner, they are flipped. The system would be even better if this was accounted for.

3.7. In this work, a task can have an arbitrary number of seeds. If a task has one seed assigned, the position of the agent and the items will always be the same. If a list of seeds is assigned, the seeds are iterated through, changing after each run. If no seed is given, a random seed is used for each run.

In the latter case there are two constraints. The first one is concerned with reproducibility. There are two kinds of seeds, the ones for generating the level and one for initializing the agent, i.e. the seed for PyTorch, for example when creating new masks or in the learning process. If there is no seed given for a task, the seeds will be generated based on the agent's seed. The second constraint is about evaluation. A list of seeds is generated at the beginning which is used by all tasks without a given seeds for all evaluations. This ensures comparability within a task and between tasks. If a task is evaluated at two different points in time, the agent was tested on the exact same level configuration. Therefore, the metrics are more comparable. For more details about the evaluation metrics, see Section 5.7.

When generating a new level, the generator first places the agent randomly. Next, the items are placed in a certain order, first the valid items for the task, then the others. The seed only determines their position. Therefore, as the seeds for all evaluation levels[2] are the same, the valid items for the tasks are all at the same positions. For example, if a particular seed places the two green items in the upper corners for the task `Pick up Green`, then the two blue items are also put in the upper corners for the task `Pick up Blue` using this seed. The same holds for `Pick up Ball` and `Pick up Box`. For the complex tasks, which have only one valid item, all target items are also at the same position, which is for this seed, one of the two upper corners. Thereby, during evaluation, all base tasks and all complex tasks are effectively tested on the same level. Figure 5 visualizes this. There the target items spawn next to the agent instead of the upper corners as described here.

4.4 Problem Relaxation

As this work is focused on combining concepts in a RL environment rather than investigating RL itself, the problem is further relaxed to better isolate the effects of the mask combinations. Therefore, a good size of the environment must be determined. To avoid unnecessary exploration, steps and field of view size, which would all also increase the computational requirements, the grid should be as small as possible. Experiments showed that if a 4×4 grid is used, the items are quite close to each other and the agent struggles navigating between them, see Figure 2. Cases in which the agent must manoeuver out of an impasse occur more often. It is worth noting that the level generator ensures that all items are reachable from the agent's starting position. Using a grid of size 5×5 is sufficient to solve the navigation problems while avoiding unnecessary exploration.

One property of the tasks in the curriculum has not been talked about in Section 4.1. All tasks can be solved without requiring memory. To make it easier for the agent to navigate through the environment, a small memory is employed. In each step, the agent is fed its last action. Thereby, the agent has a chance to escape an endless loop of turning left and right over and over.

[2]Only those without a seed given, but this is the case for most experiments.

5 Knowledge Transfer

5.1 Experiment Setup

There are two main curriculums that most of the experiments are based on. The small one investigates if masks can be combined in a productive fashion at all. It contains the tasks:

1. Pick up any item!
2. Pick up a green item!
3. Pick up a ball!
4. Pick up the green ball!

The agent is only presented with the corresponding base tasks to learn the complex tasks at the end. The big one contains all tasks, first the *Root Task*, followed by all *Base Tasks*, then all *Complex Tasks*:

1. Pick up any item!
2. Pick up a green item!
3. Pick up a blue item!
4. Pick up a ball!
5. Pick up a box!
6. Pick up the green ball!
7. Pick up the blue ball!
8. Pick up the green box!
9. Pick up the blue box!

While in the small curriculum, the agent is only presented with the corresponding base tasks to solve the complex task at the end, the big one requires the agent to identify what masks are useful for the complex tasks. Each task is trained for $300,000$ steps. The curriculum is only run through once per experiment.

In one iteration, four workers simulate 128 steps each, so 512 steps in total. Every 20 iterations, the agent is evaluated for each task on 24 runs with different seeds. The mean reward per task is calculated. The standard Minigrid reward is used:

$$1 - 0.9 \cdot \frac{\text{step\_count}}{\text{max\_steps}}$$

after the mission is completed, here, when the correct item was picked up. In all other steps, there is no reward. The quicker the agent picks up the correct item, the higher the reward. Note that a reward of 1 is hardly possible. max_steps is set to twice the number of tiles, counting the surrounding walls, which is 98. If the agent has not picked up an item by then, the run is terminated without a reward.

The network architecture is an actor critic agent with three fully connected hidden layers, each having 200 neurons. The action dimension is reduced to four: moving forward, turn left, turn right and pickup. The other actions are not necessary for the chosen tasks. As in the work of Ben-Iwhiwhu et al. (2022), a threshold of 0 for the score values is used as masking criterion. The edge-popup algorithm (Ramanujan et al., 2020) has not been tried yet. *Mask Linear Combination* as described in Ben-Iwhiwhu et al. (2022), see Section 3.3 is used to combine masks. *Balanced Linear Combination* has not been tested yet. *Mask Random Initialization*, see Section 3.7, serves as a baseline for the masking approach, see Section 5.5. The actions are chosen using a sample policy during training and a deterministic greedy policy during evaluation. As input, for each tile, color and shape are one hot encoded separately. The PPO hyperparameters, entropy weight, ratio clip and gradient clip, were approximated with a small grid search.

5.2 Evaluating Unseen Tasks

For evaluating a continual learning agent, it should be tested on all tasks seen so far, as preserving the knowledge and preventing catastrophic forgetting are the main goals. However, as this work is interested in knowledge transfer, it is also interesting how the agent performs on yet unseen tasks. For their evaluation, a linear combination of all masks trained so far is used as an estimate, in which each masks contributes equally. This is fundamentally the same as the initialization of a mask for this task if it was the next one to be trained, with the exception that no random mask is added. This would just add noise and is therefore omitted here. When implementing this, there is one problem. The mask of the currently trained task does not exist yet, but it is just a linear combination of the masks before and its random part so far. Only after the training of a task has finished, the corresponding mask is consolidated, see Figure 1.

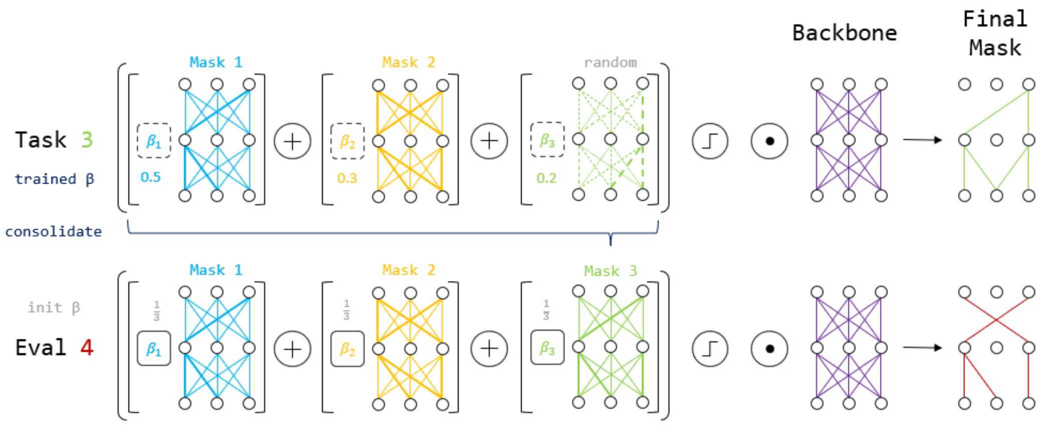


Figure 6: Visualization of the evaluation of an unseen task. Task 3 is currently trained. Trainable parameters are shown as dashed lines, fixed parameters as solid lines. Task 4 has not been seen yet and shall be evaluated by a linear combination with equal contribution of all masks, i.e. $\beta_i = \frac{1}{3}$. However, `Mask 3` does not exists yet, but is still a combination itself. The thickness of a line indicates how high it is. Note how the green `Mask 3` differs from the one above it, as the combination has been calculated.

This shall be explained using the example shown in Figure 6. Let the current task be the third task that is trained so far and the fourth task shall be evaluated. So we need $\frac{1}{3}$ of the first mask M_1, of the second mask M_2 and the third mask M_3 each. M_1 and M_2 are finished training and are already consolidated. However, M_3 is just a linear combination yet. Let the β-values of M_3 be 0.5 for M_1, 0.3 for M_2 and 0.2 for its random part. These are the β-values after softmax has been applied, so they sum up to 1. This means that M_1 contributes 50% to each score value of M_3. Therefore, M_1 contributes not just $\frac{1}{3}$ to the linear combination when evaluating the unseen task for

M_4, but also the share of the yet unconsolidated M_3 times the contribution of M_3 to it, so in total $\frac{1}{3} + \frac{1}{3} \cdot 0.5$. The same holds for M_2, but with 0.3 instead of 0.5. M_3 then contributes only $\frac{1}{3} \cdot 0.2$. Finally, those values are the desired proportions of each mask and already sum up to 1. Since afterwards a softmax function will be applied to them, the logarithm must be applied to account for that.

In the implementation of Ben-Iwhiwhu et al. (2022), the performance on unseen tasks is approximated by always using the mask of the first task. The proposed evaluation technique in this section provides an estimate closer to the real performance.

5.3 Combining Masks

To investigate the knowledge transfer potential of Modulating Masks, the approach is first tested on the small curriculum with four tasks. Figure 7 shows the average rewards per task in each evaluation round. Each task is trained for $300,000$ steps. The vertical lines visualize when a new task starts. The evaluation of unseen tasks as presented in Section 5.2 gives some intuition about the tasks. For example, during training of the first task, the agent is trained to pick up anything. By chance, in half of the cases it will pick up a ball or a green item. Therefore, we can expect the agent to achieve roughly half the performance of the first task on the corresponding tasks two (yellow) and three (green), during training of the first task. As it is slightly less, the agent apparently has a small bias towards the blue box for no obvious reason, which can be confirmed when looking at the video log. Since in the task `Pick up Green Ball` (red line), only one of the four items is valid, its approximated performance is half of the ones for task two and three. During training of the next two base tasks, the performance of the simple combination of the already trained tasks slowly increases for solving the last task (red line). After having some intuition how to read the plot, it is time to look at the most interesting part. When the training of the complex task `Pick up Green Ball` starts, it takes just a few iterations for the agent to achieve an average reward of 0.7.

Figure 8a shows the β-coefficients after softmax for the first layer. The last row indicates that the agent combines the masks for T2 (`Pick up Green`) and T3 (`Pick up Ball`) for the solution of T4 (`Pick up Green Ball`). The effect can be observed most strongly in the first layer. The higher the layer, the weaker the effect becomes, for example, Figure 8c shows the third fully connected layer.

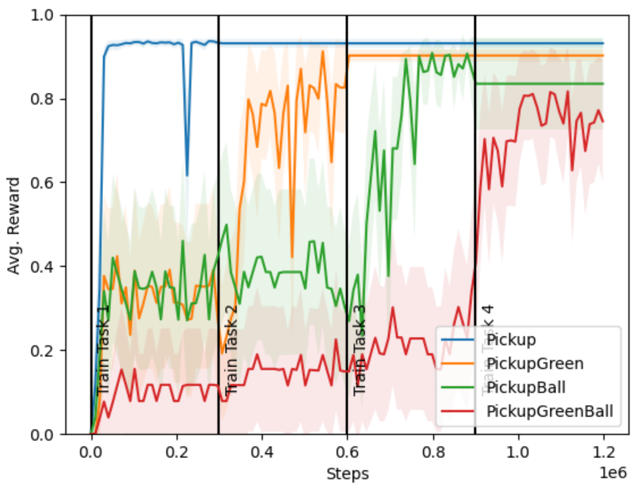
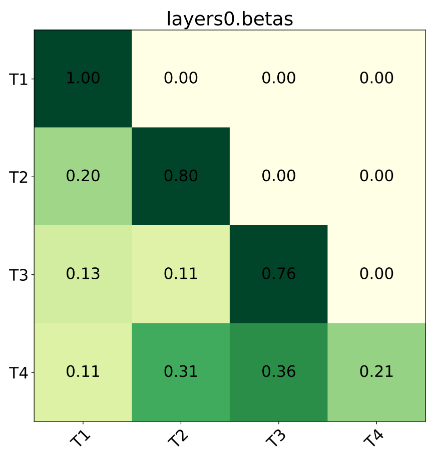


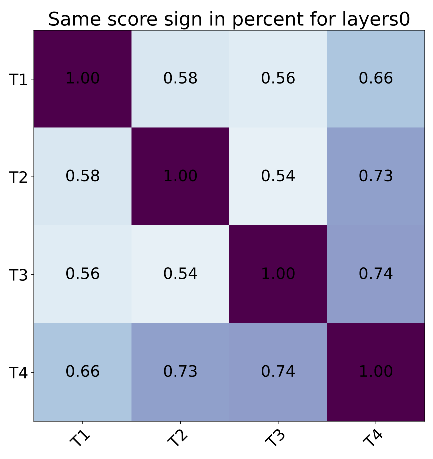
Figure 7: Plots the average reward achieved per task for the small curriculum. Every 20 iterations (512 steps each) the agent is evaluated on the same 24 levels per task. The vertical lines indicate when a new task is being trained. The order of the tasks is equivalent to the order in the legend. The shading shows the 95% confidence interval.

5.4 Mask Similarity

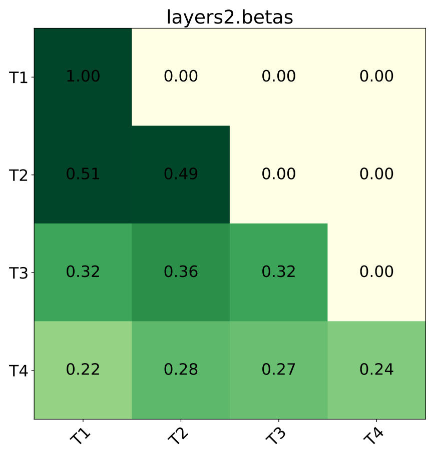
β-values show what masks were determined to be useful and which did not. However, it is not trivial to derive how they effect the resulting masks. For example, even if a mask is not useful at all, the β-value will probably never reach 0 for two reasons. First, if the contribution of the mask is small enough, the added values are too small for major changes in the resulting masks. Even if there are single counterproductive values, these can be balanced out by their counterparts in the new random mask. After that point, the algorithm might have no insentive anymore, to decrease the β-coefficients further. Second, a softmax function is applied to the raw values, which means



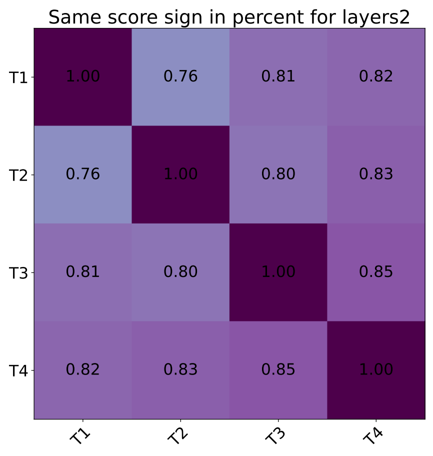
(a) β-coefficients of **first** layer



(b) similarity scores σ of **first** layer



(c) β-coefficients of **third** layer



(d) similarity scores σ of **third** layer

Figure 8: Shows the β-coefficients and the calculated similarity scores σ after training on the **small** curriculum. Each row represents one task. High β-values indicate importance of a mask when being combined for the corresponding task. β-values in one row always sum to 1. The similarity score states the percentage of corresponding score values with the same sign between two masks.

that the absolute differences in the raw β-values (before softmax) must be quite large for a coefficient to be mapped to zero.

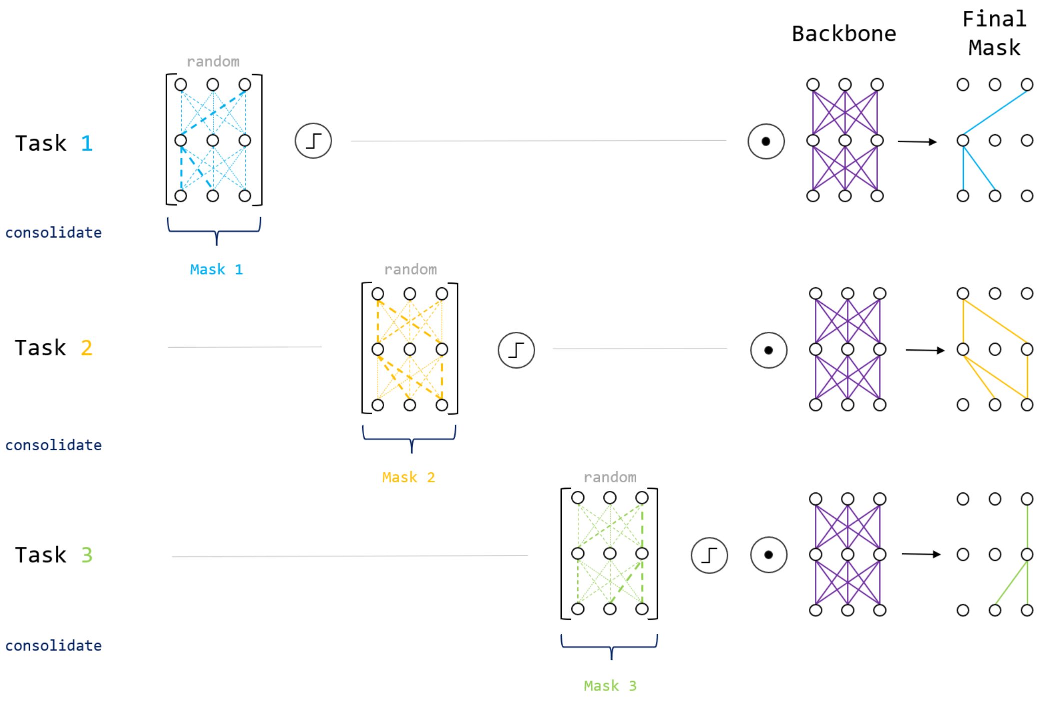
To visualize the similarities of two masks, Ben-Iwhiwhu et al. (2022) calculate the mean of the difference between corresponding score values of two masks. This metric can be used to compare which masks are more similar than others, but the values themselves are not interpretable. I argue that the score values are not important by themselves to evaluate mask similarity, as a masking criterion is applied to them, in this work, a threshold of 0. A weight is taken if its corresponding score value is positive. Therefore, when comparing two masks, the share of weights whose score values have the same sign in both masks is determined, see Figure 8b for the first layer and Figure 8d for the third layer. These values are called similarity score σ in the following. Note that since the sign of values are compared, the expected σ of two masks which have nothing in common is 0.5 just by chance. $\sigma = 0$ would mean that two masks are the exact opposite of each other.

Figure 8b shows that when the two masks for the base tasks T2 and T3 are combined for task T4, the resulting mask is quite similar. If two masks a and b have nothing in common ($\sigma_{a,b} = 0.5$), a combination c of the two cannot achieve $\sigma_{a,c} > 0.75$ and $\sigma_{b,c} > 0.75$ at the same time. Therefore, considering the similarity score between T2 and T3 is $\sigma_{2,3} = 0.54$, the mask of T4 is close to a perfect combination in the first layer, as $\sigma_{2,4} = 0.73$ and $\sigma_{3,4} = 0.74$, see Figure 8b.

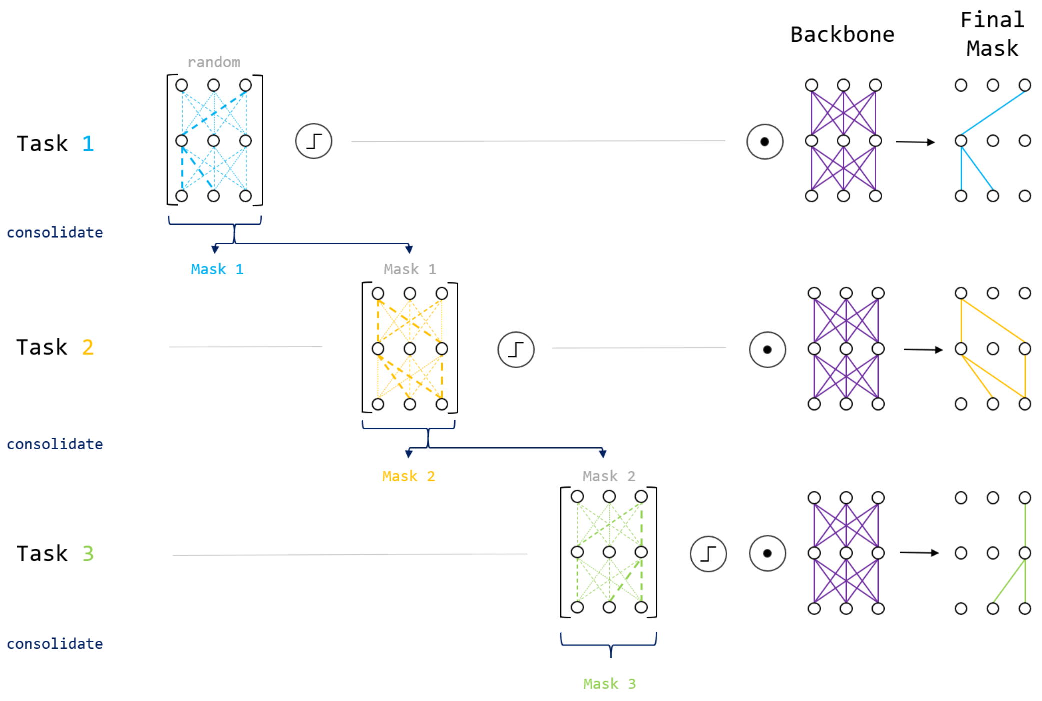
5.5 Establishing Baselines

As discussed in Section 5.3, the quick adaption of the agent to the complex task T4, `Pick up Green Ball`, as shown in Figure 7, suggests that knowledge was transferred between tasks. To determine the extend or if there is forward transfer at all, three baselines are established for this project.

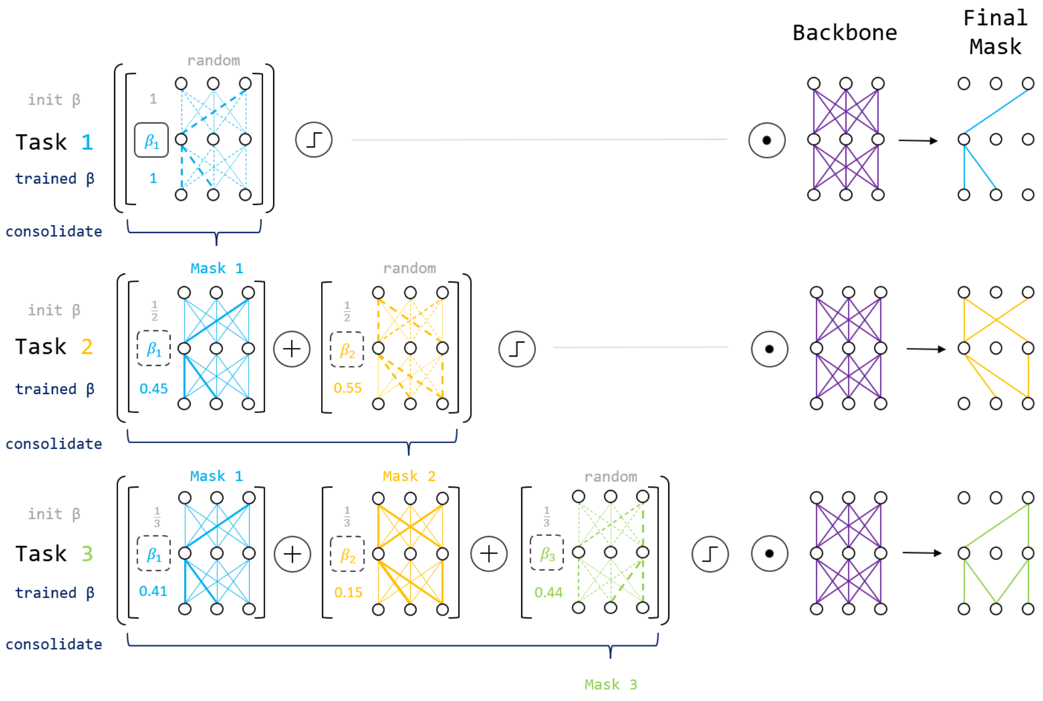
First, the approach can be compared to a general masking approach without any combinations. This is equivalent to the Mask_{RI} setting of Ben-Iwhiwhu et al. (2022), see Section 3.7. To summarize, each task is assigned a randomly initialized mask which are independent and not combined. The masking criterion, using a threshold for the score values, is kept the same for reasons of comparability. Note that, thereby, the setting is not equivalent to any approach introduced in Section 3, as there is no approach using a score



(a) Visualization of the random mask baseline



(b) Visualization of the one mask baseline



(c) Visualization of the linear combination mask algorithm, equivalent to Figure 1

Figure 9: Visualization of the baseline approaches. The main difference between the two is what happens when initializing a new mask. Initialization is represented by the color gray. The random mask approach initializes a new random mask. The one mask approach copies the mask of the task before and starts training the new task from there. Trainable parameters are shown as dashed lines, fixed parameters as solid lines. For an easier comparison of the baselines to the linear combination approach, its visualization shown in Figure 1 is repeated.

threshold with a fixed signed constant backbone network, see Table 1. For comparison with the other baselines and the linear combination approach, Figure 9a presents a visualization.

Another interesting baseline that also gives insights on the environment and the curriculum is to use the same mask for all tasks. When a new task begins, the agent just uses the current mask again. If a copy of the last mask is used and all past masks are being stored, the method still avoids catastrophic forgetting. This can be seen in Figure 9b. The motivation is that on the current small curriculum, the tasks are composed such that they built up to the complex task. When such an agent is confronted with `Pick up Green Ball` in T4, the agent starts with a solution that can already pick up balls from T3 and may even have a bias from the second task to prefer green items. A good performance of this agent might suggest that the first baseline Mask_{RI} is not very meaningful, as the tasks are too close which the Mask_{RI} agent has to learn from scratch for each task.

To compare the agent to a non-masking approach as well, the third baseline is EWC with dedicated heads for each task (Kirkpatrick et al., 2017). It was also used by Ben-Iwhiwhu et al. (2022).

Figure 10 compares the performance of the agent with all baselines on the last task, `Pick up Green Ball`. This shows that the linear combination (blue) allows forward transfer, as it takes the Mask_{RI} approach (yellow) over $200,000$ steps to catch up. However, as discussed above, the curriculum can be solved with the same performance and adoption speed by keeping the same mask for all tasks (green). EWC (red) does not perform well. A possible explanation is discussed in Section 6.5.

As the linear combination outperformed the approach without combining masks, knowledge is being transferred between tasks. However, since the approach using the same mask for all tasks was not beat, it cannot be claimed yet that the forward transfer is meaningful. Keep in mind that the one mask approach also prevents catastrophic forgetting and needs the same amount of memory, as both store one binary mask per task. Therefore, in the next step, the curriculum is extended to the large one with all nine tasks, see Section 5.1.

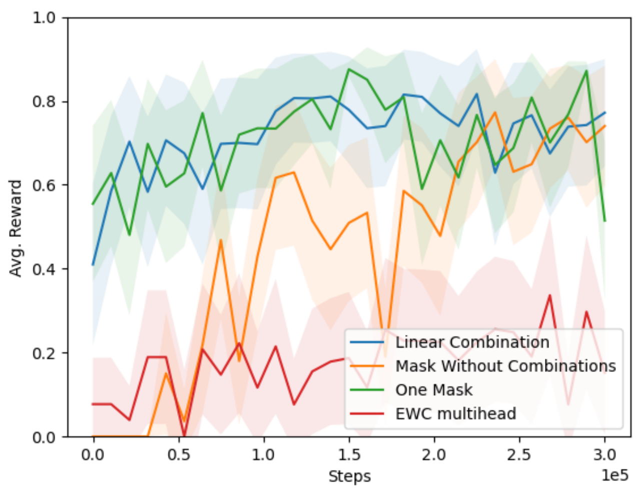


Figure 10: Plots the average reward achieved on the complex task `Pick up Green Ball` at the end of being trained on the small curriculum by the different approaches. Every 20 iterations (512 steps each) the agent is evaluated on the same 24 levels per approach. The shading shows the 95% confidence interval.

5.6 Identifying Useful Masks

So far the agent was only presented with the corresponding base tasks to the complex task. By testing the agent on the big curriculum, it also has to detect which previously trained masks are useful for the current task. Figure 11 shows the performance of the agent. The last four tasks are the complex ones. The agent adapts quite quickly to the first one (brown) and the third one (gray). When learning the task `Pick up Blue` (green) a bad adaption occurred right before training for that task ended, see the horizontal green line. This had an influence on the complex tasks involving the color blue, i.e. `Pick up Blue Ball` (pink) and `Pick up Blue Box` (olive). It took the agent longer to adapt to the same performance as on the other two complex tasks.

To check if the correct masks of the corresponding base tasks were detected, the β-values and similarity scores σ are investigated, see Figure 12. The pattern in the bottom left corner, especially in the β-values is of interest.

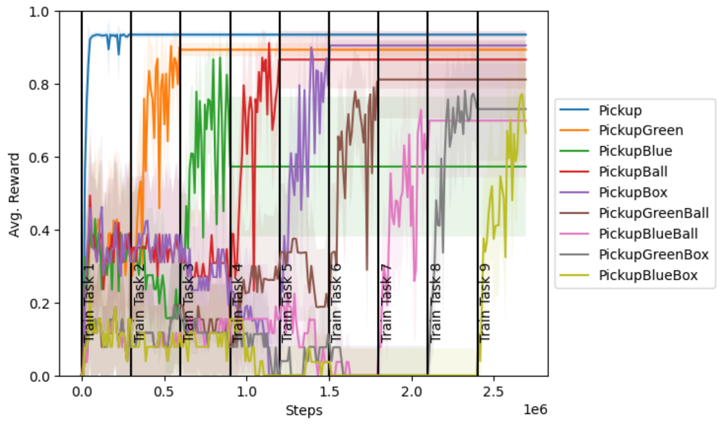
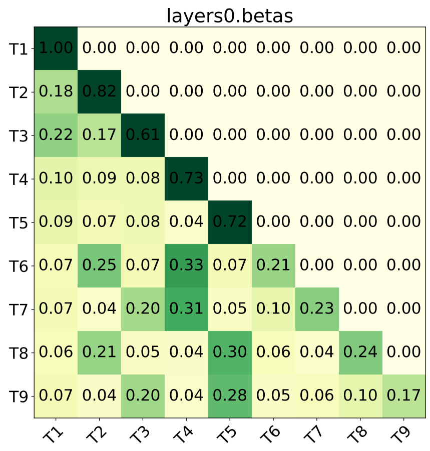
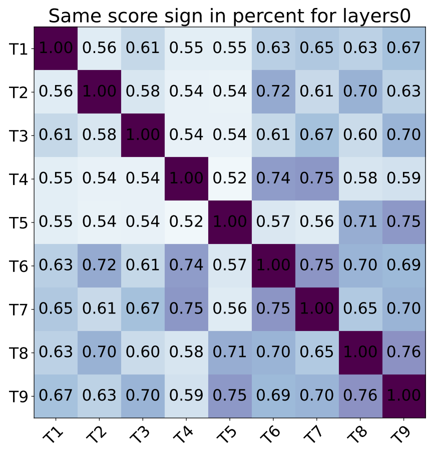


Figure 11: Plots the average reward achieved per task for the big curriculum. Every 20 iterations (512 steps each) the agent is evaluated on the same 24 levels per task. The vertical lines indicate when a new task is being trained. The order of the tasks is equivalent to the order in the legend. The shading shows the 95% confidence interval.

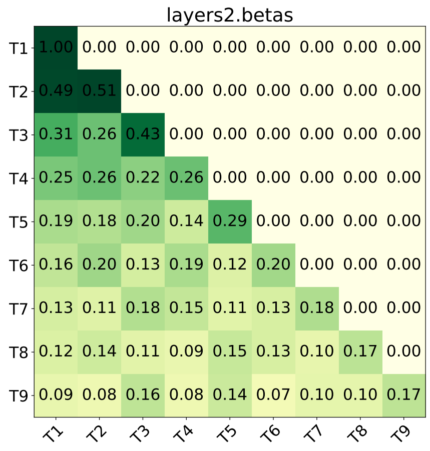
It visualizes the pattern in the curriculum. For instance, T2 and T3 correspond to *green* and *blue*, while T4 and T5 correspond to *ball* and *box*. Note how the pattern reflects the complex tasks, like T6 (*Pick up Green Ball*) has high β-coefficients for T2 and T4. Furthermore, the balance of the β-values is impressive. For example, let's look at the β-values for T9. The task is `Pick up Blue Box`. The mask with the highest coefficient is T5 (`Pick up Box`), followed by T3 (`Pick up Blue`). There is quite a difference between the two, but keep in mind that `Pick up Blue` was the task the agent made a bad adaptation right before training ended, see Figure 11. Therefore, this is expected. T8 (`Pick up Green Box`) also has a notable contribution. Both T8 and T9 are looking for a box. The lowest β-coefficients belong to T1 (`Pick up Green`), T4 (`Pick up Ball`) and T6 (`Pick up Green Ball`), all having nothing in common with the current task T9 (`Pick up Blue Box`). The similarity scores in Figure 12b confirm the findings. Therefore, the linear combination is able to identify useful masks on the fly during training. Note that the pattern can be observed best in the first layer, see Figure 12a and 12b. The higher the layer, the less the pattern stands out and the more similar all masks become. For example, Figure 12c and 12d show the β-



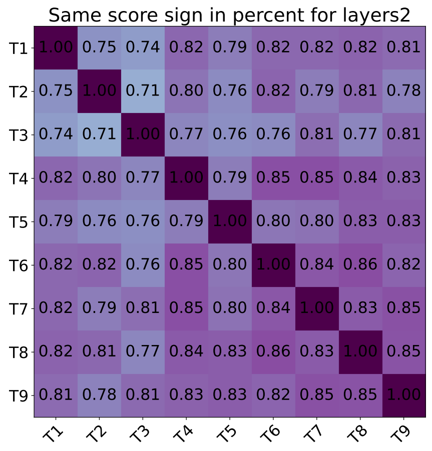
(a) β-coefficients of **first** layer



(b) similarity scores σ of **first** layer



(c) β-coefficients of **third** layer



(d) similarity scores σ of **third** layer

Figure 12: Shows the β-coefficients and the calculated similarity scores σ after training on the **big** curriculum. Each row represents one task. High β-values indicate importance of a mask when being combined for the corresponding task. β-values in one row always sum to 1. The similarity score states the percentage of corresponding score values with the same sign between two masks.

values and σ-values for the third layer. The reason and implications of this are discussed in Section 6.3.

5.7 Measuring Forward Transfer

The degree of forward transfer is determined by comparing the agent to the baseline approaches introduced in Section 5.5. Figure 13 shows the average performance of the different approaches for the four complex tasks.

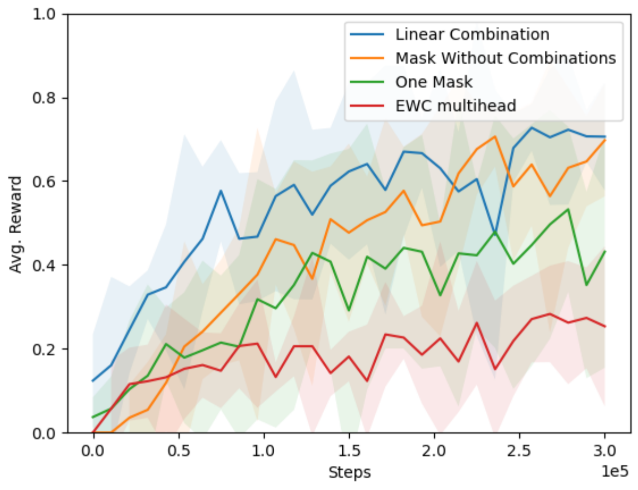
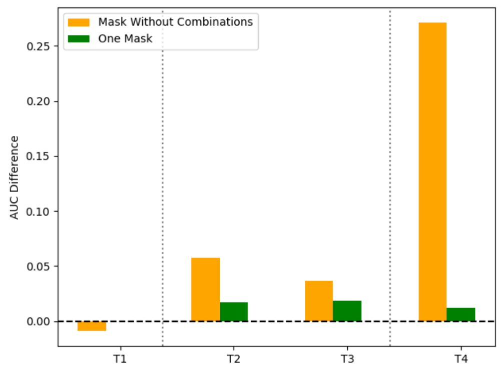
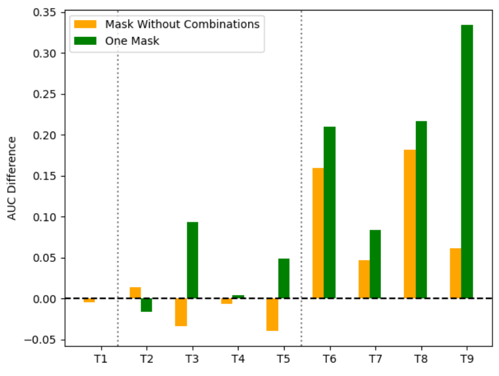


Figure 13: Plots the average reward achieved on the four complex tasks at the end of being trained on the big curriculum by the different approaches. Every 20 iterations (512 steps each) the agent is evaluated on the same 24 levels per approach. The shading shows the 95% confidence interval.

On the small curriculum, the first performance value of the linear combination approach on the complex task is 0.41, see Figure 10. Now, the agent's first performance is 0.12 (blue). This is expected, as the agent has to identify useful masks first. The approach still adapts quicker to the complex tasks than learning a new mask from scratch (yellow). Furthermore, the approach using only one mask (green) is outperformed in the big curriculum. In the small curriculum, it performs as well as the linear combination, see Figure 10. Therefore, now the tasks, or at least their order, are different enough such that relying on the most recent mask is not an effective strategy anymore.



(a) **small** curriculum



(b) **big** curriculum

Figure 14: Shows the average difference of the area under curve between the linear combination approach and different baselines. A value of 0.1 indicates, that the linear combination approach scores on average 0.1 reward more than the compared baseline at any point during training of the corresponding task on that very task. The dotted vertical lines separate root task (`Pick up Anything`), base tasks and complex tasks.

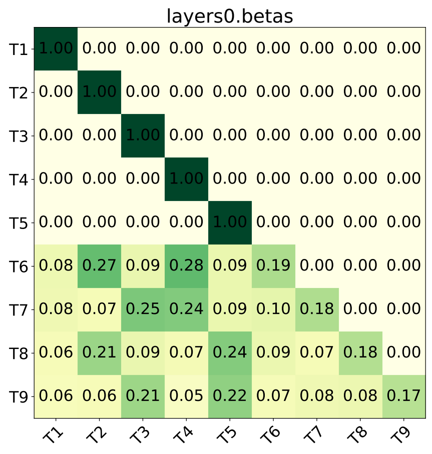
Again, EWC does not perform well. See Section 6.5 for a possible explanation. It is not that important that the random mask approach (yellow) catches up after about $200,000$ steps. It is already known that we can train a RL agent to solve Minigrid tasks. As this work is interested in investigating learning concepts and reusing them, we are more focused on adapting quickly, in the long run, ideally in a few-shot or if the task descriptions are known to the agent, even in a zero-shot setting.

So far, most of the analysis was qualitative. To measure forward transfer, the average area under curve for each approach and task is calculated by averaging the corresponding performance values. As EWC could not achieve a comparable performance to the linear combination, it is omitted in the further evaluation. Figure 14a shows the difference in the AUC of the linear combination and the remaining approaches of the small curriculum, Figure 14b analogously for the big curriculum. A positive value indicates the lead of the linear combination approach. It can be interpreted as the average reward the linear combination agent scores higher at any run than the compared baseline during training of that task.

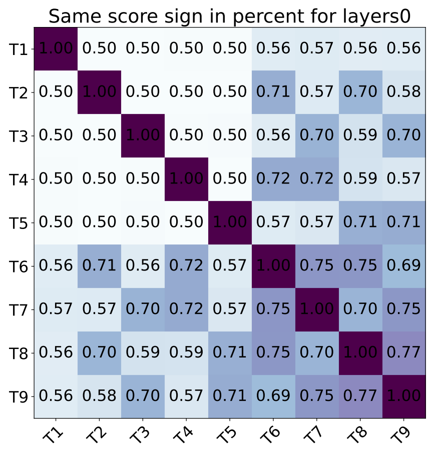
In Figure 14a, the large yellow bar indicates the transferred knowledge in the linear combination as discussed in Section 5.5. On average, the linear combinations scores 0.27 more reward per run. For comparison, the highest performance score on that task was 0.81 (average of 24 runs with linear combination). The results shown in Figure 14b can be explained as follows. In the base tasks, the approaches differ especially in T3 and T5. The reason is that these tasks are more different to their predecessors than the other base tasks. T3 (`Pick up Blue`) is the opposite to T2 (`Pick up Green`). This can also be observed in the suggested task embedding space, see Section 4.2. While T2 and T3 point to opposite directions, T2 and T3 are perpendicular to each other. The random mask baseline is not affected by this. The linear combination approach is, but can decrease the contribution of T2 by reducing the corresponding β-value. The one mask approach on the other hand has to adjust the score values themselves starting with a counterproductive bias. Therefore, the positive green bar cannot be seen as forward transfer by the linear combination but rather as a better adaption capability. A similar argument can be made for T5. Note that the bad adaption in T3 (`Pick up Blue`) by the linear combination agent, see Section 5.6, has barely any influence on the metrics for the base tasks, as this affects only one value averaged for calculating the metrics for T3. However, it is the reason for the smaller lead of the linear combination agent on tasks T7 (`Pick up Blue Ball`) and T9 (`Pick up Blue Box`) with respect to the random mask agent. If the bad adaption has not happened, the forward transfer advantage of the linear combination agent might be even larger. However, those bad adaptations can happen in the current approach and are, therefore, a current limitation. On the other complex tasks, a forward transfer of up to 0.18 could be achieved compared to the random mask agent on T8 and a of up to 0.33 compared to the one mask agent on T9.

5.8 Independent Masks

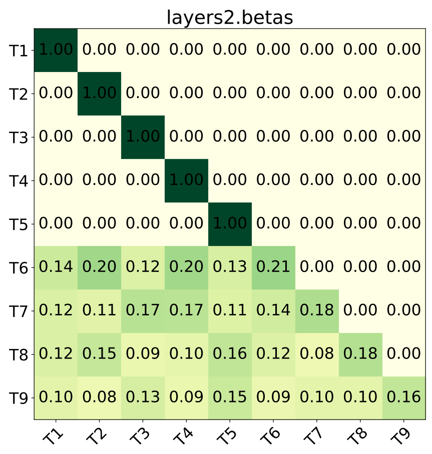
One not yet mentioned hypothesis of this work is that masks must be trained on each other such that they are combinable. The idea is that even though some masks are not useful to solve a task, the fact that they are used in the linear combination at the beginning of training a task, ensures that the resulting masks is pushed towards a solution that shares certain structures. Thereby, the possibility of combining masks shall be improved.



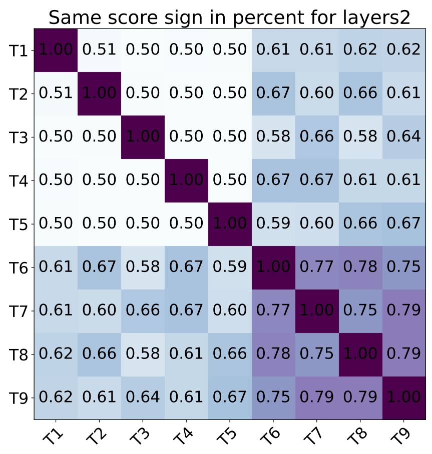
(a) β-coefficients of **first** layer



(b) similarity scores σ of **first** layer



(c) β-coefficients of **third** layer



(d) similarity scores σ of **third** layer

Figure 15: Shows the β-coefficients and the calculated similarity scores σ after training on the big curriculum, in which the root and base tasks were trained **independently**. Each row represents one task. High β-values indicate importance of a mask when being combined for the corresponding task. β-values in one row always sum to 1. The similarity score states the percentage of corresponding score values with the same sign between two masks.

However, when looking at the similarity scores between the base tasks T2 - T5 in Figure 12b, most of them are just slightly over 0.5. Note that $\sigma = 0.5$ is the baseline similarity by chance which indicates that the masks have nothing in common, see Section 5.4. Nevertheless, the combination of the masks works well. This questions the aforementioned hypothesis. In the next experiment, the root and base tasks are independently trained. This means that there is no linear combination of masks for those tasks similar to the baseline approach without combination using Mask_{RI}.

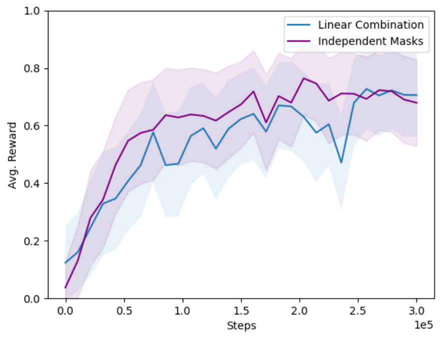


Figure 16: Plots the average reward achieved on the four complex tasks at the end of being trained on the big curriculum for each approach. The shading shows the 95% confidence interval.

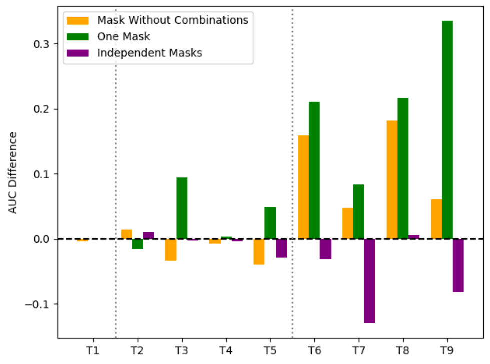


Figure 17: Shows the average difference of the area under curve between the linear combination approach and different baselines. Extends Figure 14b by the independent mask experiment.

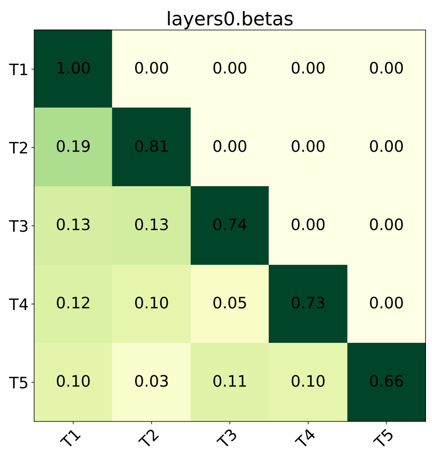
Figure 15 shows the corresponding β- and σ-values. The $\beta = 0$ in the little triangle for the base tasks T2 - T5 is the result of them being independently trained. Yet, the same pattern in the complex tasks arises, proving that the correct masks are still identified and combined. The pattern is also clearly visible in the σ-scores, see Figure 15b. This demonstrates that even independently trained masks can be linearly combined in this setting. When comparing the results to the regular linear combination agent, Figure 16 suggests that the independent approach performs even better. However, more experiments are needed to support that claim, as the majority comes from the lead on T7 and T9, see Figure 17, the ones affected by the bad adaptation of the linear combination agent.

5.9 Extracting Concepts

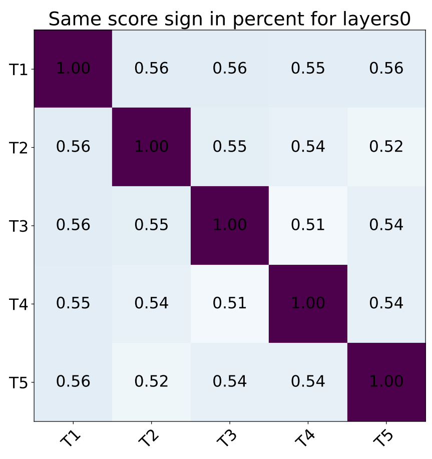
So far, the agent got to learn the corresponding base tasks before being confronted with the complex tasks. Their masks contribute most to the resulting masks of the complex ones. However, some complex tasks also correlate, for instance, `Pick up Green Ball` and `Pick up Blue Ball` both share the shape *ball*. This can also be observed in the suggested task embeddings, as they share one axis but differ in the other one, see Section 4.2. Therefore, it shall be investigated if the linear combination algorithm as proposed is able to extract concepts such as *ball* from existing masks to achieve forward transfer. To test this capability, the agent is presented with a new curriculum consisting of the root task and the complex tasks, thereby, skipping the base tasks:

1. Pick up any item!
2. Pick up the green ball!
3. Pick up the blue ball!
4. Pick up the green box!
5. Pick up the blue box!

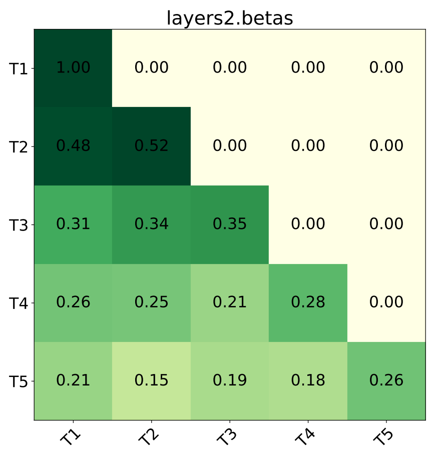
The β-values depicted in Figure 18a do not show any masks that are identified as useful for a combination. The agent is able to detect masks that are not useful at all. For example T2 (`Pick up Green Ball`) and T5 (`Pick up Blue Box`) are opposing tasks. The algorithm recognized this and assigned $\beta_{5,2} = 0.03$. Same holds for tasks T3 and T4. The similarity scores in Figure 18b indicate that there is barely any combination taking place. However, let's compare the similarity scores to a previous experiment, in which the base tasks were trained independently, see Section 5.8. The masks for the base tasks have nothing in common ($\sigma = 0.5$) in Figure 15b. However, when combined, even masks from opposing complex tasks like T6 and T9 as well as T7 and T8 have a similarity score of $\sigma_{9,6} = 0.69$ and $\sigma_{8,7} = 0.70$, see Figure 15b. This happens although the base tasks of the respective opposing complex task are hardly included in the combination, see β-values in Figure 15a. No good explanation has been found for this behaviour yet.



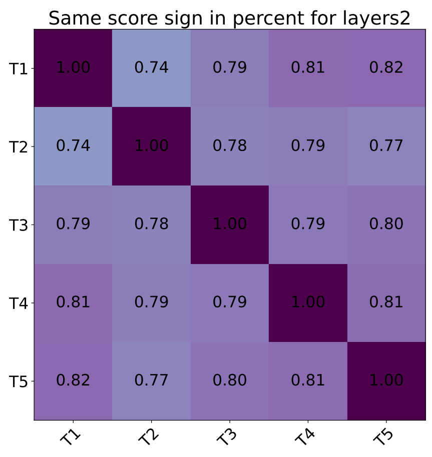
(a) β-coefficients of **first** layer



(b) similarity scores σ of **first** layer



(c) β-coefficients of **third** layer



(d) similarity scores σ of **third** layer

Figure 18: Shows the β-coefficients and the calculated similarity scores σ after training on a curriculum containing **only complex tasks** and the root task. Each row represents one task. High β-values indicate importance of a mask when being combined for the corresponding task. β-values in one row always sum to 1. The similarity score states the percentage of corresponding score values with the same sign between two masks.

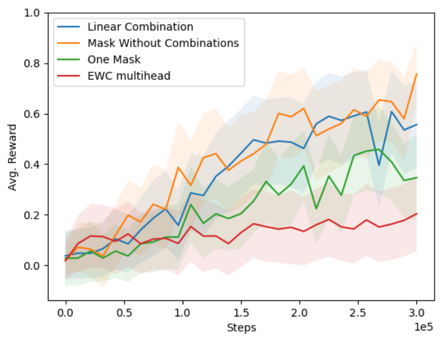


Figure 19: Plots the average reward achieved on the four complex tasks at the end of being trained on the root and complex tasks for each approach. The shading shows the 95% confidence interval.

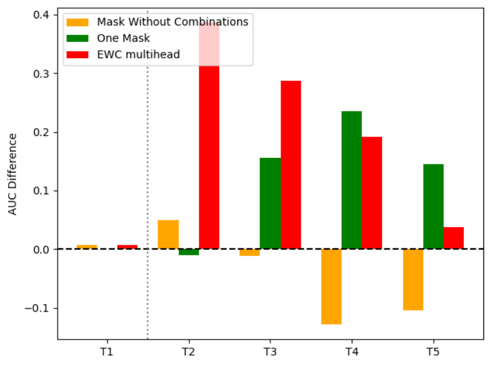


Figure 20: Shows the average difference of the area under curve between the linear combination approach and different baselines.

When comparing the approach to the baselines, the one mask approach and EWC are still outperformed, see Figure 19. However, as the random mask approach, which does not combine masks, performs better as the linear combination approach especially when more previous masks are being present in T4 and T5, see Figure 20, there is no forward transfer anymore. EWC, while being outperformed in the beginning, catches up with more tasks being present. On the one hand the linear comb approach is getting worse. This can be derived from the change in the random mask approach. As all complex tasks are equally difficult to solve, the performance of the approach without combining masks (orange) is expected to perform equally on all of them. As it expands its lead with respect to the linear combination approach, the latter one must get worse with more previous masks to handle. On the other hand, EWC is also getting better over time. Probably the weights have been consolidated in a productive fashion and introducing a new head for a task starts to bear fruits.

The experiments in this section show that the linear combination approach as proposed cannot extract concepts from existing masks.

6 Explanation Attempt

Some key results of the experiments are:

- Useful masks can be identified and combined in the big curriculum, Section 5.6.
- Masks cannot be effectively combined when presented only with the complex tasks, Section 5.9.
- The combinations are more selective in lower layers, i.e. the β-values of previous masks differ more, Section 5.6.

This section tries to bring those findings together by giving an explanation attempt for the questions: Why does combining masks work in some scenarios but not in slightly altered ones? When does it work? What happens when masks are learned and combined? It is important to keep in mind that the explanations in this section simplify a lot and are just hypotheses. They have yet to be tested in further experiments and analyses which are not in scope of this project.

6.1 Connecting Paths

Let's say the agent shall learn the task `Pick up Green`. What the agent has to do first is to learn what input neurons indicate *green*. Next, those are put together to identify an area where a green item is found in the field of view. A direction can be determined and is forwarded to the action layer. Finally, an action must be derived from that information. Blue items are not relevant for that scenario besides that they cannot be walked over. Of course, during training, this is not an iterative process but learned in parallel. Let's simplify this further. There is a certain neuron that flags if the tile two steps in front of the agent is green or not. If it is green, the agent should decide to walk straight forward. To know this, there must be a path through the layers that transports this information from the input neuron to the action layer. Probably, there will be multiple of those paths. Some of those paths may be shared with other neurons. For example, the neuron indicating *green* for the tile three steps ahead of the agent could use the same paths. Regardless which of the two contain a green item, the action should be in both cases *move forward.*

6.2 Reusing Lanes

If the agent now learns another task like `Pick up Blue`, the mask of the previous task is linearly combined with the new random mask for this task. In the inputs, the color of a tile is one-hot encoded. This means that the neurons which flag a green tile are different from the ones flagging a blue tile. Therefore, the first layer of the mask for task `Pick up Green` is useless for the new task or even counterproductive, as we do not want to forward the location of green items but blue ones now. This can be observed in the β-values. In Figure 12a, $\beta_{3,2} = 0.17$, which is the corresponding β-value of those two tasks in the curriculum in the first layer. This value was initialized with 0.33, so it has been reduced to roughly half its initial contribution. However, the paths that lead from the first layer to the action layer could be reused. Instead of being connected to the green input neurons, they must be reconnected with the blue input neurons. This will not be possible one by one. For sake of simplicity, let's say that a path uses one weight per layer to connect an input neuron with the action layer. To transmit the information that the target color has been found, a positive value is transmitted via this path. However, if the weight in the backbone network between the blue input neuron of the same tile and the neuron of the next layer in the path is negative instead of positive as it was with the green input neuron, this exact same path cannot be taken. This can also be seen in the β-values. The second layer is shown in Figure 12c, in which $\beta_{3,2} = 0.26$.

6.3 Higher Layer Similarity

Furthermore, all masks are more similar to each other in higher layers, see Figure 12d, than in lower layers, see Figure 12b. To explain this, let's say there is a path in the mask for `Pick up Green`, that connects a green input neuron to the action head. When learning `Pick up Blue`, this path shall be reused to transfer the information if the same tile contains a blue item now. To connect the new neuron in the first layer, the weight of the backbone network connecting the neuron to the one in the path in the next layer must be the same as the one connecting the green input neuron to the path. The likelihood of this is 50%. Otherwise, the blue input neuron must first be connected to another neuron in the next layer and hop on the path in the following layer. There will be an intermediate neuron that provides weights with the necessary sign. Once on the path, there is no reason not to reuse the knowledge how to built those paths to the action layer, which has been learned in previous masks. Therefore, it gets more and more likely that the new neuron is connected to the path the higher the layer is and reusing previous masks becomes more lucrative.

6.4 Fusing Routes

Now, let's say the agent is confronted with the task `Pick up Green Ball` and has learned the tasks `Pick up Green` and `Pick up Ball`. To solve the new task, the agent must learn an AND-gate of the two previous tasks. When the masks of the two base tasks are combined, the paths used by both of them will receive high score values and will be taken. In the first layer, the two masks for the base tasks are quite different, $\sigma_{4,2} = 0.54$ in Figure 12b. This is expected, as they focus on feeding forward different input neurons, those corresponding to *green* and those corresponding to *ball*. Therefore, in the first layer, it is important that both masks have a high contribution to the resulting mask to keep both. Figure 12a shows that $\beta_{6,2} = 0.25$ and $\beta_{6,4} = 0.33$ for task T6 (`Pick up Green Ball`), where the initial values were $\beta = \frac{1}{6}$. The other base tasks like `Pick up Blue` or `Pick up Box` learned paths that transfer the wrong information for the current task. Therefore, those paths must be interrupted as early as possible. In lower layers it is not just important to take the masks of the useful base tasks, but especially not taking the masks of the other tasks, i.e. assigning them a low β-value. On higher layers on the other side, the counterproductive paths of the other base tasks were quite likely already interrupted for the most part. Therefore, it is less important to avoid them and the β-values can be more balanced, see Figure 12c. The reasoning in this section also holds for the scenario, in which the base tasks were trained independently.

6.5 Dedicated Heads

The given intuition about the environment may also explain the bad performance of EWC. In a masking approaches with and without linear combination, one mask is only concerned with creating paths for the properties the current task asks for, such as *green* or `ball`. When a new task is presented, the linear combination approach can exchange the used masks in the first layers. The random mask baseline is not concerned with previous tasks anyway. EWC, on the other hand, tries to maintain the important weights in the first layers for solving the previous tasks, while creating a dedicated head at higher layers for new tasks. There are several disadvantages for EWC given the curriculum and the environment. First, there must be paths for all properties, even the ones not asked for in the current task, up until the heads. Furthermore, those paths must be kept indefinitely, to avoid catastrophic forgetting. Finally, the tasks are trained in sequence without seeing a task twice, for $300,000$ steps each. The only things that can be put aside during the time a task is not trained is its dedicated head, which is on a high layer.

6.6 Splitting Tracks

The final question is, why extracting concepts does not work in the curriculum which skips the base tasks, see Section 5.9. As concepts like *green* and *ball* were never trained independently, their paths are intertwined from the very beginning. Before, they could be addressed individually using β-values, as they were in different masks. This is not possible here. Therefore, the paths of all of them must be interrupted, hence, all β-values are quite low. As some paths of `Pick up Green Ball` do at least not hinder the performance on `Pick up Blue Ball`, it is less important to interrupt them as with opposing tasks, such as `Pickup Green Ball` and `Pick up Blue Box`, which share no properties. They were pushed down by a lot, see $\beta_{4,3} = 0.05$ and $\beta_{5,2} = 0.03$ in Figure 18a.

7 Implementation

The code is based on the repository provided by Ben-Iwhiwhu et al. (2022), but was fundamentally reorganized such that new experiments can be set up with ease. The original implementation put most code belonging to an experiment into a few big functions. These contained a lot of redundancies and were hard to read. Therefore, code of the functions was organized in dedicated modules, for instance, `my_config.py`, `my_logger.py`, `my_eval.py`, `my_analysis.py`, `my_utils.py` and others. A selection of them are described in more detail.

To distinguish what I have contributed to the code base from what other people have coded before me, I put most of my code in the python modules in the root folder starting with `my_...`, whereas pre-existing libraries are kept in folders. This shall simplify the work of the assessors of this project by increasing transparency.

7.1 Experiment Configuration

Rather than creating a config object individually for each experiment, the config module pursues a more structured approach. Three classes of config properties are determined and put into designated functions:

1. `_build_config_base`: General RL properties and organizational properties that are the same for all experiments, e.g. logging paths, the discount factor, evaluation properties, ...
2. `_build_minigrid_config_base`: Environment specific properties like the task functions
3. `build_minigrid_config`: CL-preservation technique specific properties such as a masking-approach or EWC

This structure allows to extend the system, for instance, by new environments. In that case, only a new function on the second layer must be written containing only the environment specific properties. Avoiding the redundancies in that way comes with the known advantages. For example, if properties of the evaluation shall be changed, this must only be adjusted once and inconsistent behaviour between runs can be mitigated.

Experiment specific properties may change more often. Therefore, they are not put in code but rather kept in dedicated json files in the `env_configs` folder. The name of the config json file will also be the name of the experiment used for identifying it in the log folder. The most important properties in those files are the curriculum represented by the task names, the number of training steps for task, the CL-preservation technique and its properties. For example, `new_task_mask` allows to select the `linear_comb` masking algorithm as well as the `random` mask baseline.

7.2 Training Evaluation

All code snippets regarding evaluation are put in the dedicated `my_eval.py` module to make the main loop more readable. To approximate the performance of the agent on unseen tasks, Ben-Iwhiwhu et al. (2022) used the first mask for all tasks. In Section 5.2, I propose a solution that mimics what would happen when a new mask would be initialized for an unseen task right now by equally combining all masks trained so far. This change must be done in the agent. The module `my_agent.py` contains a superclass of the agent used by Ben-Iwhiwhu et al. (2022) with my extensions for the evaluation of unseen tasks. This mainly involves calculating β-values as described in Section 5.2.

Additionally, I implemented a new evaluation feature, the video log, allowing qualitative analyses. All evaluation runs are recorded and saved. Corresponding properties like frames per second and the flag to switch recording on or off are put in the config module. Visualizing how the agent behaves can be helpful to avoid optimizing for metrics but getting an intuition what is learned. Specifically, this was very useful when designing the environment.

7.3 Agent Analysis

While the evaluation module is active during training, the analysis module contains analytical functionalities after training. The code is mostly reused from Ben-Iwhiwhu et al. (2022) but, as the rest, reorganized to avoid redundancies. This module can be executed manually on an existing log file, for example, to compare it to another run. The mask similarity function is adjusted to fit the new proposal described in Section 5.4. Analysis that is concerned with calculating forward transfer is written in the `my_comparisons.ipynb` notebook.

7.4 Specifying Masks

During the project, I experimented with implementing custom masks, which are not covered in this report. However, I extended the actor critic network to allow an easy exchange of the used masks. The config automatically uses corresponding superclasses I created in `my_masks.py`. For instance, this allows to specify the desired mask type such as *threshold* based or *sparse* (top k%) in the config json file. The implementation ensures that the masks are exchanged in all layers of the body and the heads of the actor-critic architecture. Own mask algorithms can be included here.

7.5 Custom Tasks

All code that is related to integrating an environment into the RL-agent, has been organized in the `my_tasks.py` module. I extended the code such that it is possible to put in the same task more than once and allow custom seeds. Before, each task had to have a dedicated seed. Thereby, the items in the environment would always be at the same positions. In Section 3.7, I argue that this makes it harder to assess what the agent has truly learned. Therefore, my implementation allows a task to have a single seed, a list of seeds which is iterated through and no seed. This is specified alongside the task names in the config json file. A list can be given using square brackets. If no seed is given, represented as any character or string not an integer or list, a new seed is generated after each run, i.e. until the agent has picked something up or the maximum number of steps has been reached. The seeds for evaluation runs are predetermined and the same for all tasks, see Section 4.3.

The module `my_minigrid_green_blue.py` hosts the code for the environment and is built on the Minigrid framework. It is inspired by implementations of pre-existing Minigrid levels and is, therefore, quite compact. In particular functionality to ensure that all items are reachable could be reused. All tasks can be generated from one generic custom `Pickup` class. When registering the tasks to Minigrid such that their names can be found by the gymnasium framework, the color and shape of the target item(s) are specified. The name used here is the same that allows to identify the task in the config json file.

7.6 Running Experiments

Through the effort of modularizing the existing code, the `my_experiments.py` module is quite clean. For instance, the main loop is quite now quite short and readable:

```python
for task_idx, task_info in enumerate(tasks_info):
    log_new_task_starts(config, task_idx, task_info)

    prepare_agent_for_task(agent, task_info)

    while True:
        # train step
        dict_logs = agent.iteration()
        iteration += 1

        # logging
        log_iteration(agent, iteration)

        # evaluate agent
        eval_agent(agent, tasks_info, iteration)

        # check if task training has been completed
        if is_task_training_complete(agent, task_idx):
            break
```

Listing 1: Shows the main loop for each experiment. Before each task the agent is notified to prepare for the task, for example, creating a new mask. The endless loop consists of executing an iteration, logging, and checking if the agent shall be evaluated.

To create a new experiment, the base function, i.e. `learn_green_blue`, is copied and extended with the experiment specific code. For example, for the baseline using the same mask for all tasks, this was achieved by changing the agent's method that maps a task label to the mask index. The inserted lambda function always returns the same mask. Note that while creating the same effect for evaluating and comparison, this does not store a copy of the previous masks as suggested in Section 5.5. This is sufficient for being a baseline though. For the experiment investigating independently trained masks, the β-values for the current task are adjusted such that the new random masks contributes 100% in the linear combination. Next, the β-coefficients are set as untrainable. The number of independently trained tasks is a property that can be specified in the json config file, but is only necessary if this experiment is run.

The code base contains an `environment.yml` file to setup a conda virtual environment which allows executing the code. Alternatively, the file `my_requirements.txt` also includes all necessary dependencies. To execute an experiment, adjust the path to the config json file and select the function of the desired experiment at the end of the `my_experiments.py` file. Then run the module with the CUDA prefix:

```
$ conda env create -f environment.yml
$ conda activate mask_env
$ CUDA_VISIBLE_DEVICES=1 python3 my_experiments.py
```

8 Further Improvements

In this section, I present aspects that can further be improved. This includes how the linear combination algorithm may be enhanced based on the results of the experiments in Section 5 as well as a critical reflection on the project itself.

8.1 Beta Behavior

In the combination of masks, the β-values have a significant influence on the success of the resulting mask, as they determine which masks are used to what degree. This section discusses some thoughts how the behaviour of the β-coefficients could be altered to increase their effectiveness based on observations in the experiments.

When the algorithm detects masks that are not useful for solving the current task, the corresponding β-values are reduced. It could be investigated how quickly this happens. For example, Figure 12a shows the β-values for the big curriculum experiment. In the last task T9, the base tasks T2 and T4 are counterproductive for solving the task. Therefore, their coefficients are pushed down to 0.04. How many steps or iterations does it take until they reach this small value. Furthermore, since the masks are clearly not helpful, should the algorithm be able to set the coefficients to 0? Since a softmax is applied to the learnable raw coefficient, this is hardly possible so far. $\beta = 0$ would also mean that the agent cannot reactivate the mask anymore, as there will not be any gradients anymore. Probably, those masks have no direct impact on the result anyway if the β-value is small enough. This requires that score values cannot explode but are bound, such that a mask cannot have some extraordinary large score values. Wortsman et al. (2020) can detect the correct mask in a one-shot or few-shot setting. However, they investigate a supervised classification problem, see Section 3.2, whereas this work is concerned with a RL-agent. Therefore, their work can hardly be compared with the problem setting here.

Section 5.9 investigated if concepts can be extracted. While this was not the case, the agent could detect what masks were not useful at all, for example `Pick up Green Ball` when solving `Pick up Blue Box`. Could β-values also be negative to create *inhibitory masks*. Instead of adding score values, they would be subtracted. Two questions must be answered. First, what mechanism allows β-values to switch between positive and negative values. Second, does subtracting score values lead to a productive outcome?

8.2 Sparse Masks

Like Ben-Iwhiwhu et al. (2022), a threshold based masking criterion is used in this work. As the initial score values are randomized and the used threshold for score values is 0, roughly half of the score values will be positive and, therefore, half of the weights will be taken. In an algorithm like the *edge-popup*, the top k% of the weights are selected. Ramanujan et al. (2020) found out that a k around 50% seemed optimal. They propose a combinatorial argument for this, as the most number of subnetworks can be created that way, see Section 3.1.

It can be argued that masks should be as sparse as possible as long as it does not have a negative impact on performance. If masks only contain weights necessary for their task, this may reduces side effects when combining them. This might be more relevant when the system is scaled up to contain more combinable concepts. As concepts can be of different complexity, setting a fixed value might not be the best approach rather than implementing an incentive for the agent to learn masks as sparse as possible by itself. One initial approach could be an automatic decay for score values. In any training step all score values are reduced, or at least the ones being above the masking threshold. If a weight is important for solving the task, gradient decent will push it up again. Otherwise, the weight will fall below the masking threshold eventually.

8.3 Statistical Significance

There are two kinds of seeds. One determines the random initialization for the application, the other one is concerned with creating environments. The latter one has discussed a lot, for example, how reshuffling the items in the environments after each run helps to learn concepts rather trajectories and how the randomness can be controlled for comparability in evaluation, see Section 4.3. However, the first seed was barely talked about. In fact, all experiments in this project were run with the initialization seed of 42.

However, it is necessary to run the same experiments with multiple seeds to allow statements that are significant. One example that illustrates this is the bad adaption of the linear combination agent for a base task that had implications for the effectiveness of combining masks, see Section 5.6. It is crucial to investigate how often this occurs. For instance, when calculating forward transfer, this effect barely concerns the random mask baseline but has a significant influence on the linear combination approach, see Section 5.7.

Running the experiments with multiple seeds would most likely reduce the uncertainty of most confidence intervals in the plots and allow to calculate confidence intervals in the first place for the forward transfer bar plots, i.e. Figure 14a, 14b, 17 and 20. For investigating the objectives of this project, the used methodology is sufficient. The bad adaptation for example even gave some insights into the functionality of combining masks, as there was a comparison between well and badly trained base tasks in the combinations. However, for publishing the results, running more experiments will become necessary.

8.4 Beyond Scope

There are a few other open questions that are not in scope of this project but shall be mentioned. This work focused on combining mask and the resulting forward transfer. Each task was only seen once in a curriculum. If a task is seen again, training could continue just using its dedicated mask. However, is it also possible to achieve backward transfer by linearly combining masks of tasks seen later? Moreover, the performance plots do not show a consistent growth, but are quite volatile. In the long run, a more stable training curve would be desirable. Furthermore, it has been described that the environment has been designed such that the tasks have interesting properties to each other. Section 4.2 showed how these can be embedded in an easy fashion. How these can be used in the learning process, maybe in the outer loop of a meta-learning, see Section 2.3, is out of scope of this project. Finally, while the intuition about masks presented in Section 6 does explain the findings of the experiments, the structure of masks must further be examined to validate the reasoning. Since this would fill up a project on its own, it cannot be covered in this report anymore.

9 Conclusion

To summarize the report, the research questions stated in Section 1.2 are examined to recapitulate what was found out and to what degree the objectives of the project were met.

Can masks be trained such that they resemble meaningful knowledge or concepts? Since in the current implementation a mask represents a task, the environment and its tasks must be carefully created. To allow masks that resemble interpretable and meaningful concepts, a custom Minigrid environment was designed, see Section 4. When learning complex tasks in the big curriculum, the masks that contributed most are the corresponding base tasks, see Figure 12a. The effect is strongest in the first layer. On higher layers, I proposed that masks probably work differently, see Section 6.3. Therefore, through the elaborate thoughts put into designing the environment, I could show that masks can resemble meaningful and interpretable concepts at least on lower layers.

Can such masks be created such that they are combinable with each other? The small curriculum served as a prove of concept that masks can be combined. The masks of the base tasks made a disproportionately large contribution to the resulting masks. To avoid relying on the β-values alone and check if they are informative, I proposed a new way to determine mask similarity. The σ-scores showed that masks are combined as expected, see Figure 8a and 8b.

Is the algorithm able to detect useful masks for the current task among the already trained ones to combine them? The emerging pattern in the β-values indicates that the algorithm can detect useful masks for a task from a larger selection and combine them, see Figure 12a. Going beyond, I investigated under what circumstances mask combination works. I could show that masks of base tasks can even be combined when trained independently, see Figure 15, but subconcepts cannot be extracted from complex tasks, see Figure 18.

Does combining those masks allow the agent to adapt quicker to new tasks, i.e. is there forward transfer? To determine forward transfer, I created a suite of baselines, with each having a different strength to compare the agent to. The random mask starts from scratch for each task and is, therefore, neither positively nor negatively influenced by other tasks.

The one mask approach exceeds when similar tasks are learned in sequence. Finally, EWC is employed as a non-masking approach. I could show that the linear combination approach adapts quicker to the complex tasks than all baselines in the big curriculum. To make stronger statements about forward transfer, the experiments must be re-run with several seeds, see Section 8.3.

What can be inferred from the results with respect to how masks and their combination work? In an attempt to bring the findings of the experiments together, I propose an explanation of the inner mechanisms of masks and their combination in Section 6. The purpose of masks is simplified to selecting valuable information from the input layer and forwarding it via paths to the action layer. The findings of the experiments can be explained with this model by thinking about how those paths can be disconnected, reconnected, merged and inextricably intertwined. While the effort to prove it by analysing the resulting masks in detail is beyond the scope of this project, I can unite all findings of the experiments.

The findings of this work are novel and relevant for the field and should be published. Two aspects should be improved to uphold a higher scientific standard. First, the performance of the learning plots should be more stable, and the experiments should be re-run with multiple seeds to allow stronger statements. Second, the presented intuition about the functionality of masks in Section 6 was not the main focus of this work and is, therefore, just a plausible hypothesis yet. If it can be confirmed in further analyses, it would round up a possible publication.

In this thesis, I have presented a novel view on masks which increases learning speed by incremental concept learning and points out several aspects how to further improve the approach. The achieved forward transfer shows potential that further developments in the area of concept learning based on neural architectures can lead to the creation of more intelligent RL-agents. The insights how masks and their combinations work can build the foundation for a deeper understanding of the functionality of masks, leading to more effectively designed systems.

Bibliography

Ben-Iwhiwhu, E., Nath, S., Pilly, P. K., Kolouri, S., and Soltoggio, A. (2022). Lifelong reinforcement learning with modulating masks. *arXiv preprint arXiv:2212.11110*.

Bengio, S., Bengio, Y., Cloutier, J., and Gescei, J. (2013a). On the optimization of a synaptic learning rule. In *Optimality in Biological and Artificial Networks?*, pages 281–303. Routledge.

Bengio, Y., Léonard, N., and Courville, A. (2013b). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Chevalier-Boisvert, M., Dai, B., Towers, M., de Lazcano, R., Willems, L., Lahlou, S., Pal, S., Castro, P. S., and Terry, J. (2023). Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831.

Courbariaux, M., Bengio, Y., and David, J.-P. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28.

De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385.

Denil, M., Shakibi, B., Dinh, L., Ranzato, M., and De Freitas, N. (2013). Predicting parameters in deep learning. *Advances in neural information processing systems*, 26.

Devin, C., Gupta, A., Darrell, T., Abbeel, P., and Levine, S. (2017). Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 2169–2176. IEEE.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., and Pontil, M. (2018). Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1568–1577. PMLR.

Frankle, J. and Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135.

Goldblum, M., Fowl, L., and Goldstein, T. (2020). Adversarially robust few-shot learning: A meta-learning approach. *Advances in Neural Information Processing Systems*, 33:17886–17895.

Griffiths, T. L., Callaway, F., Chang, M. B., Grant, E., Krueger, P. M., and Lieder, F. (2019). Doing more with less: meta-reasoning and meta-learning in humans and machines. *Current Opinion in Behavioral Sciences*, 29:24–30.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. (2021). Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169.

Houthooft, R., Chen, Y., Isola, P., Stadie, B., Wolski, F., Jonathan Ho, O., and Abbeel, P. (2018). Evolved policy gradients. *Advances in Neural Information Processing Systems*, 31.

Kang, H., Mina, R. J. L., Madjid, S. R. H., Yoon, J., Hasegawa-Johnson, M., Hwang, S. J., and Yoo, C. D. (2022). Forget-free continual learning with winning subnetworks. In *International Conference on Machine Learning*, pages 10734–10750. PMLR.

Kessler, S., Parker-Holder, J., Ball, P., Zohren, S., and Roberts, S. J. (2022). Same state, different task: Continual reinforcement learning without interference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7143–7151.

Khetarpal, K., Riemer, M., Rish, I., and Precup, D. (2022). Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476.

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

Li, C., Farkhoor, H., Liu, R., and Yosinski, J. (2018). Measuring the intrinsic dimension of objective landscapes. *arXiv preprint arXiv:1804.08838*.

Li, Y., Yang, Y., Zhou, W., and Hospedales, T. (2019). Feature-critic networks for heterogeneous domain generalization. In *International Conference on Machine Learning*, pages 3915–3924. PMLR.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Mallya, A., Davis, D., and Lazebnik, S. (2018). Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 67–82.

Mallya, A. and Lazebnik, S. (2018). Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., and Rastegari, M. (2020). What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11893–11902.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. (2017). icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010.

Ring, M. B. (1997). Child: A first step towards continual learning. *Machine Learning*, 28(1):77–104.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30.

Soltoggio, A., Ben-Iwhiwhu, E., Peridis, C., Ladosz, P., Dick, J., Pilly, P. K., and Kolouri, S. (2023). The configurable tree graph (ct-graph): measurable problems in partially observable and distal reward environments for lifelong reinforcement learning. *arXiv preprint arXiv:2302.10887*.

Wortsman, M., Ramanujan, V., Liu, R., Kembhavi, A., Rastegari, M., Yosinski, J., and Farhadi, A. (2020). Supermasks in superposition. *Advances in Neural Information Processing Systems*, 33:15173–15184.

Yoon, J., Yang, E., Lee, J., and Hwang, S. J. (2018). Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*.

Zhou, H., Lan, J., Liu, R., and Yosinski, J. (2019). Deconstructing lottery tickets: Zeros, signs, and the supermask. *Advances in neural information processing systems*, 32.