

Mix & Match – Agent Curricula for Reinforcement Learning

Wojciech Marian Czarnecki^{*1} Siddhant M. Jayakumar^{*1} Max Jaderberg¹ Leonard Hasenclever¹
Yee Whye Teh¹ Simon Osindero¹ Nicolas Heess¹ Razvan Pascanu¹

Abstract

We introduce Mix & Match (M&M) – a training framework designed to facilitate rapid and effective learning in RL agents, especially those that would be too slow or too challenging to train otherwise. The key innovation is a procedure that allows us to automatically form a *curriculum over agents*. Through such a curriculum we can progressively train more complex agents by, effectively, bootstrapping from solutions found by simpler agents. In contradistinction to typical curriculum learning approaches, we do not gradually modify the tasks or environments presented, but instead use a process to gradually alter how the policy is represented internally. We show the broad applicability of our method by demonstrating significant performance gains in three different experimental setups: (1) We train an agent able to control more than 700 actions in a challenging 3D first-person task; using our method to progress through an action-space curriculum we achieve both faster training and better final performance than one obtains using traditional methods. (2) We further show that M&M can be used successfully to progress through a curriculum of architectural variants defining an agent’s internal state. (3) Finally, we illustrate how a variant of our method can be used to improve agent performance in a multitask setting.

1 Introduction

The field of deep reinforcement learning has seen significant advances in the recent past. Innovations in environment design have led to a range of exciting, challenging and visually rich 3D worlds (e.g. Beattie et al., 2016; Kempka

^{*}Equal contribution ¹DeepMind, London, UK. Correspondence to: W Czarnecki <lejlot@google.com>, S Jayakumar <sidmj@google.com>.

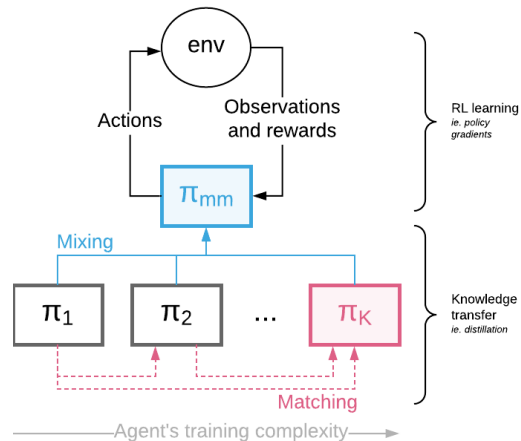


Figure 1. Scheme of Mix & Match– each box represents a policy. The blue π_{mm} is the control policy, optimised with the true RL objective. White boxes represent policies used for the curriculum, while the red policy is the final agent.

et al., 2016; Brockman et al., 2016). These have in turn led to the development of more complex agent architectures and necessitated massively parallelisable policy gradient and Q-learning based RL algorithms (e.g. Mnih et al., 2016; Espeholt et al., 2018). While the efficacy of these methods is undeniable, the problems we consider increasingly require more powerful models, complex action spaces and challenging training regimes for learning to occur.

Curriculum learning is a powerful instrument in the deep learning toolbox (e.g. Bengio et al., 2009; Graves et al., 2017). In a typical setup, one trains a network sequentially on related problems of increasing difficulty, with the end goal of maximizing final performance on a desired task. However, such task-oriented curricula pose some practical difficulties for reinforcement learning. For instance, they require a certain understanding of and control over the generation process of the environment, such that simpler variants of the task can be constructed. And in situations where this is possible, it is not always obvious how to construct useful curricula – simple intuitions from learning in humans do not always apply to neural networks. Recently (e.g. Sutskever & Zaremba, 2014; Graves et al., 2017) proposes randomised or automated curricula to cir-

cumvent some of these issues with some success. In this paper, instead of curricula over task variants we consider an alternate formulation – namely a curriculum over variants of *agents*. We are interested in training a single final agent, and in order to do so we leverage a series of intermediate agents that differ structurally in the way in which they construct their policies (Fig. 1). Agents in such curricula are not arranged according to architectural complexity, but rather training complexity. While these complexity measures are often aligned, they are sometimes orthogonal (*e.g.* it is often faster to train a complex model on two distinct tasks, than a simpler model on them jointly). In contrast to a curriculum over tasks, our approach can be applied to a wide variety of problems where we do not have the ability to modify the underlying task specification or design. However, in domains where traditional curricula are applicable, these two methods can be easily combined.

The primary contribution of this work is thus to motivate and provide a principled approach for training with curricula over agents.

Mix & Match: An overview

In the Mix & Match framework, we treat multiple agents of increasing learning complexity as one *M&M agent*, which acts with a *mixture* of policies from its constituent agents (Fig. 1). Consequently it can be seen as an ensemble or a mixture of experts agent, which is used solely for purpose of training. Additionally, knowledge transfer (*i.e.* distillation) is used such that we encourage the complex agents to *match* the simpler ones early on. The mixing coefficient is controlled such that ultimately only the complex, target agent is used for generating experience. Note that we consider the complexity of an agent not just in terms of the depth or size of its network (see Section 4.2), but with reference to the difficulty in training it from scratch (see Section 4.1 and 4.3). We also note that while analogous techniques to ours can be applied to train mixtures of experts/policies (*i.e.* maximising performance across agents), this is not the focus of the present research; here our focus is to train a final target agent.

Training with the Mix & Match framework confers several potential benefits – for instance performance maximisation (either with respect to score or data efficiency), or enabling effective learning in otherwise hard-to-train models. And with reference to this last point, M&M might be particularly beneficial in settings where real world constraints (inference speed, memory) demand the use of certain specific final models.

2 Related work

Curriculum learning is a long standing idea in machine learning, with mentions as early as the work of Elman (Elman, 1993). In its simplest form, pretraining and finetuning is a form of curriculum, widely explored (*e.g.* Simonyan & Zisserman, 2014). More explicitly, several works look at the importance of a curriculum for neural networks (*e.g.* Bengio et al., 2009). In many works, this focus is on constructing a sequence of tasks of increasing difficulty. More recent work (Graves et al., 2017; Sutskever & Zaremba, 2014) however looks at automating task selection or employing a mixture of difficulties at each stage in training. We propose to extend this idea and apply it instead to training agents in curricula – keeping in spirit recent ideas of mixtures of tasks (here, models).

The recent work on Net2Net (Chen et al., 2016) proposes a technique to increase the capacity of a model without changing the underlying function it represents. In order to achieve this, the architectures have to be supersets/subsets of one another and be capable of expressing identity mappings. Follow-up work (Wei et al., 2016) extends these ideas further. Both approaches can be seen as implicitly constructing a form of curriculum over the architecture, as a narrow architecture is first trained, then morphed into a wider one.

Related to this idea is the concept of knowledge transfer or distillation (Hinton et al., 2015; Ba & Caruana, 2014) – a technique for transferring the functional behaviour of a network into a different model, regardless of the structure of the target or source networks. While initially proposed for model compression (Bucilu et al., 2006; Ba & Caruana, 2014), in (Parisotto et al., 2016; Rusu et al., 2016) distillation is used to compress multiple distinct policies into a single one. Distral (Teh et al., 2017) instead focuses on learning independent policies, which use co-distilled centralised agent as a communication channel.

Our work borrows and unifies several of these threads with a focus on online, end-to-end training of model curricula from scratch.

3 Method details

We first introduce some notation to more precisely describe our framework. Let us assume we are given a sequence of trainable agents¹ (with corresponding policies π_1, \dots, π_K , each parametrised with some $\theta_i \subset \theta$ – which can share some parameters) ordered according to the complexity of

¹ For simplicity of notation we are omitting time dependence of all random variables, however we do consider a time-extended setting. In particular, when we talk about policy of agent i we refer to this agent policy at given time (which will change in the next step).

interest (*i.e.* π_1 can be a policy using a tiny neural network while π_K the very complex one). The aim is to train π_K , while all remaining agents are there to induce faster/easier learning. Furthermore, let us introduce the categorical random variable $c \sim \text{Cat}(1, \dots, K|\alpha)$ (with probability mass function $p(c = i) = \alpha_i$) which will be used to select a policy at a given time:

$$\pi_{\text{mm}}(a|s) = \sum_{i=1}^K \alpha_i \pi_i(a|s).$$

The point of Mix & Match is to allow curriculum learning, consequently we need the probability mass function (pmf) of c to be changed over time. Initially the pmf should have $\alpha_1 = 1$ and near the end of training $\alpha_K = 1$ thus allowing the curriculum of policies from simple π_1 to the target one π_K . Note, that α_i has to be adjusted in order to control learning dynamics and to maximise the whole learning performance, rather than immediate increase. Consequently it should be trained in a way which maximises long lasting increase of performance (as opposed to gradient based optimisation which tends to be greedy and focus on immediate rewards).

We further note that mixing of policies is necessary but not sufficient to obtain *curriculum learning* – even though (for non dirac delta like c) gradients always flows through multiple policies, there is nothing causing them to actually share knowledge. In fact, this sort of mixture of experts is inherently competitive rather than cooperative (Jacobs et al., 1991). In order to address this issue we propose using a distillation-like cost D , which will align the policies together.

$$\mathcal{L}_{\text{mm}}(\theta) = \sum_{i,j=1}^K D(\pi_i(\cdot|\cdot, \theta_i), \pi_j(\cdot|\cdot, \theta_j), i, j, \alpha).$$

The specific implementation of the above cost will vary from application to application. In the following sections we look at a few possible approaches.

The final optimisation problem we consider is just a weighted sum of the original \mathcal{L}_{RL} loss (*i.e.* A3C (Mnih et al., 2016)), applied to the control policy π_{mm} and the knowledge transfer loss:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{RL}}(\theta|\pi_{\text{mm}}) + \lambda \mathcal{L}_{\text{mm}}(\theta).$$

We now describe in more detail each module required to implement Mix & Match, starting with policy mixing, knowledge transfer and finally α adjustment.

3.1 Policy mixing

There are two equivalent views of the proposed policy mixing element – one can either think about having a categor-

ical selector random variable c described before, or an explicit mixing of the policy. The expected gradients of both are the same:

$$\begin{aligned} \mathbb{E}_{i \sim c} [\nabla_{\theta} \pi_i(a|s, \theta)] &= \sum_{i=1}^K \alpha_i \nabla_{\theta} \pi_i(a|s, \theta) = \\ &= \nabla_{\theta} \sum_{i=1}^K \alpha_i \pi_i(a|s, \theta) = \nabla_{\theta} \pi_{\text{mm}}(a|s, \theta), \end{aligned}$$

however, if one implements the method by actually sampling from c and then executing a given policy, the resulting single gradient update will be different than the one obtained from explicitly mixing the policy. From this perspective it can be seen as a Monte Carlo estimate of the mixed policy, thus for the sake of variance reduction we use explicit mixing in all experiments in this paper.

3.2 Knowledge transfer

For simplicity we consider the case of $K = 2$, but all following methods have a natural extension to an arbitrary number of policies. Also, for notational simplicity we drop the dependence of the losses or policies on θ when it is obvious from context.

Consider the problem of ensuring that final policy π_2 matches the simpler policy π_1 , while having access to samples from the control policy, π_{mm} . For simplicity, we define our M&M loss over the trajectories directly, similarly to the unsupervised auxiliary losses (Jaderberg et al., 2017b; Mirowski et al., 2017), thus we put:

$$\mathcal{L}_{\text{mm}}(\theta) = \frac{1 - \alpha}{|S|} \sum_{s \in S} \sum_{t=1}^{|s|} D_{\text{KL}}(\pi_1(\cdot|s_t) \| \pi_2(\cdot|s_t)), \quad (1)$$

and trajectories ($s \in S$) are sampled from the control policy. The $1 - \alpha$ term is introduced so that the distillation cost disappears when we switch to π_2 .² This is similar to the original policy distillation (Rusu et al., 2016), however here the control policy is mixture of π_2 (the student) and π_1 (the teacher).

One can use a memory buffer to store S and ensure that targets do not drift too much (Ross et al., 2011). In such a setting, under reasonable assumptions one can prove the convergence of π_2 to π_1 given enough capacity and experience.

Remark 1. *Lets assume we are given a set of N trajectories from some predefined mix $\pi_{\text{mm}} = (1 - \alpha)\pi_1 + \alpha\pi_2$ for any fixed $\alpha \in (0, 1)$ and a big enough neural network with softmax output layer as π_2 . Then in the limit as $N \rightarrow \infty$, the minimisation of Eq. 1 converges to π_1 if the optimiser*

²It can also be justified as a distillation of mixture policy, see Appendix for derivation.

used is globally convergent when minimising cross entropy over a finite dataset.

Proof. Given in the appendix. \square

In practice we have found that minimising this loss in an online manner (*i.e.* using the current on-policy trajectory as the only sample for Eq. (1)) works well in the considered applications.

3.3 Adjusting α through training

An important component of the proposed method is how to set values of α through time. For simplicity let us again consider the case of $K = 2$, where one needs just a single α (as c now comes from Bernoulli distribution) which we treat as a function of time t .

Hand crafted schedule Probably the most common approach is to define a schedule by hand. Unfortunately, this requires per problem fitting, which might be time consuming. Furthermore while designing an annealing schedule is simple (given that we provide enough flat regions so that RL training is stable), following this path might miss the opportunity to learn a better policy using non-monotonic switches in α .

Online hyperparameter tuning Since α changes through time one cannot use typical hyperparameter tuning techniques (like grid search or simple Bayesian optimisation) as the space of possible values is exponential in number of timesteps ($\alpha = (\alpha^{(1)}, \dots, \alpha^{(T)}) \in \Delta_{K-1}^T$, where Δ_k denotes a k dimensional simplex). One possible technique to achieve this goal is the recently proposed Population Based Training (Jaderberg et al., 2017a) (PBT) which keeps a population of agents, trained in parallel, in order to optimise hyperparameters through time (without the need of ever reinitialising networks). For the rest of the paper we rely on using PBT for α adaptation, and discuss it in more detail in the next section.

3.4 Population based training and M&M

Population based training (PBT) is a recently proposed learning scheme, which performs online adaptation of hyperparameters in conjunction with parameter optimisation and a form of online model selection. As opposed to many classical hyperparameter optimisation schemes– the ability of PBT to modify hyperparameters throughout a single training run makes it is possible to discover powerful adaptive strategies *e.g.* auto-tuned learning rate annealing schedules.

The core idea is to train a population of agents in parallel, which periodically query each other to check how well they

are doing relative to others. Badly performing agents copy the weights (neural network parameters) of stronger agents and perform local modifications of their hyperparameters. This way poorly performing agents are used to explore the hyperparameters space.

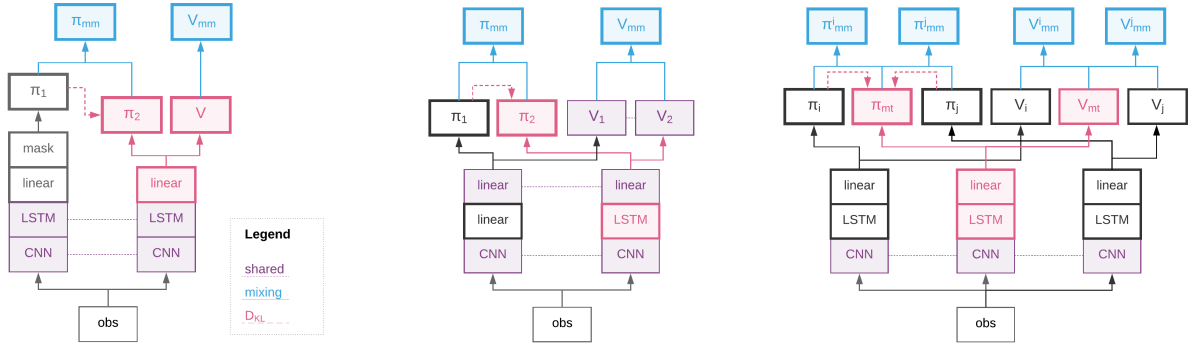
From a technical perspective, one needs to define two functions – `eval` which measures how strong a current agent is and `explore` which defines how to perturb the hyperparameters. As a result of such runs we obtain agents maximising the `eval` function. Note that when we refer to an *agent* in the PBT context we actually mean the M&M agent, which is already a mixture of constituent agents.

We propose to use one of the two schemes, depending on the characteristics of the problem we are interested in. If the models considered have a clear benefit (in terms of performance) of switching from simple to the more complex model, then all one needs to do is provide `eval` with performance (*i.e.* reward over k episodes) of the mixed policy. For an `explore` function for α we randomly add or subtract a fixed value (truncating between 0 and 1). Thus, once there is a significant benefit of switching to more complex one – PBT will do it automatically. On the other hand, often we want to switch from an unconstrained architecture to some specific, heavily constrained one (where there may not be an obvious benefit in performance from switching). In such setting, as is the case when training a multitask policy from constituent single-task policies, we can make `eval` an independent evaluation job which only looks at performance of an agent with $\alpha_K = 1$. This way we directly optimise for the final performance of the model of interest, but at the cost of additional evaluations needed for PBT.

4 Experiments

We now test and analyse our method on three sets of RL experiments. We train all agents with a form of batched actor critic with an off policy correction (Espenholt et al., 2018) using DeepMind Lab (Beattie et al., 2016) as an environment suite. This environment offers a range of challenging 3D, first-person view based tasks (see, appendix) for RL agents. Agents perceive 96×72 pixel based RGB observations and can move, rotate, jump and tag built-in bots.

We start by demonstrating how M&M can be used to scale to a large and complex action space. We follow this with results of scaling complexities of the agent architecture and finally on a problem of learning a multitask policy. In all following sections we do not force α to approach 1, instead we initialise it around 0 and analyse its adaptation through time. Unless otherwise stated, the `eval` function returns averaged rewards from last 30 episodes of the control policy. Note, that even though in the experimental sections we



(a) M&M for action spaces progression (b) M&M for architecture progression (c) M&M for multitask progression

Figure 2. Schemes of various settings M&M can be applied to, explored in this paper. Violet nodes represent modules that are shared between π_i , grey ones are separate modules for the helper agents and red ones – modules which are unique to the final agent. Blue nodes are the ones that are exposed to the environment – control policy(ies) and value function(s).

use $K = 2$, the actual curriculum goes through potentially infinitely many agents being a result of mixing between π_1 and π_2 . Further technical details and descriptions of all tasks are provided in Appendix.

4.1 Curricula over number of actions used

DeepMind Lab provides the agent with a complex action space, represented as a 6 dimensional vector. Two of these action groups are very high resolution (rotation and looking up/down actions), allowing up to 1025 values. The remaining four groups are low resolution actions, such as the ternary action of moving forward, backward or not moving at all, shooting or not shooting etc. If naively approached this leads to around $4 \cdot 10^{13}$ possible actions at each timestep.

Even though this action space is defined by the environment, practitioners usually use an extremely reduced subset of available actions (Mnih et al., 2016; Espeholt et al., 2018; Jaderberg et al., 2017b; Mirowski et al., 2017) – from 9 to 23 preselected ones. When referring to action spaces we mean the subset of possible actions used for which the agent’s policy provides a non zero probability. Smaller action spaces significantly simplify the exploration problem and introduce a strong inductive bias into the action space definition. However, having such a tiny subset of possible movements can be harmful for the final performance of the agent. Consequently, we apply M&M to this problem of scaling action spaces. We use 9 actions to construct π_1 , the simple policy (called *Small action space*). This is only used to guide learning of our final agent – which in this case uses 756 actions – these are all possible combinations of available actions in the environment (when limiting the agent to 5 values of rotation about the z-axis, and 3 values about the x-axis). Similarly to the research in continuous control

using diagonal Gaussian distributions (Heess et al., 2017) we use a factorised policy (and thus assume conditional independence given state) to represent the joint distribution $\pi_2(a_1, a_2, \dots, a_6|s) := \prod_{j=1}^6 \hat{\pi}_j(a_j|s)$, which we refer to as *Big action space*. In order to be able to mix these two policies we map π_1 actions onto the corresponding ones in the action space of π_2 (which is a strict superset of π_1).

We use a simple architecture of a convolutional network followed by an LSTM, analogous to previous works in this domain (Jaderberg et al., 2017b). For M&M we share all elements of two agents apart from the final linear transformation into the policy/value functions (Fig. 2a). Full details of the experimental hyper-parameters can be found in the appendix, and on each figure we show the average over 3 runs for each result.

We see that the small action space leads to faster learning but hampers final performance as compared to the big action space (Fig. 3 and Fig. 4). Mix & Match applied to this setting gets the best of both worlds – it learns fast, and not only matches, but surpasses the final performance of the big action space. One possible explanation for this increase is the better exploration afforded by the small action space early on, which allows agents to exploit fully their flexibility of movement.

We further compare two variants of our method. We first investigate using *M&M (Shared Head)* – in this approach, we share weights in the final layer for those actions that are common to both policies. This is achieved by masking the factorised policy π_2 and renormalising accordingly. We further consider a variant of our distillation cost – when computing the KL between π_1 and π_2 one can also mask this loss such that π_2 is not penalised for assigning non-zero probabilities to the actions outside the π_1 support – *M&M (Masked KL)*. Consistently across tested levels, both

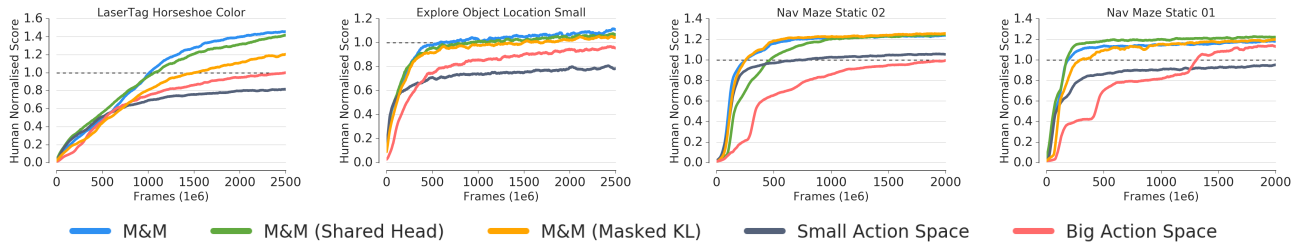


Figure 3. Comparison of M&M and its variations to baselines applied to the problem of scaling to complex action spaces. Each figure shows the results for a single DMLab level. Each curve shows the mean over 3 populations, each consisting of 10 agents each, with random seeds and hyperparameters as described in the appendix. M&M represents the formulation with mixing and a KL distillation cost. The masked KL version only incorporates a KL cost on those actions present in both policies, while the shared head variant shares weights for these common actions directly. All variants of M&M outperform the baselines in data efficiency and performance.

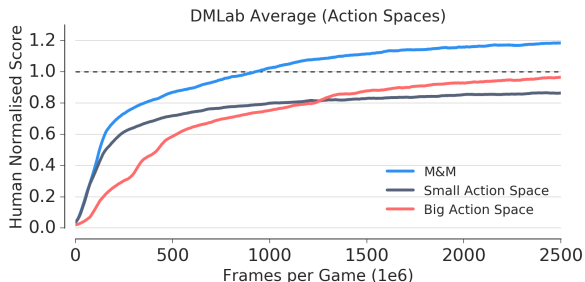


Figure 4. Human normalised score through training in **action spaces** experiments. Averaged across levels (for per level curves see Fig. 3).

shared Head and Masked KL approaches achieve comparable or worse performance than the original formulation. It is worth noting however, that if M&M were to be applied to a non-factorised complex action space, the Masked KL might prove beneficial, as it would then be the only signal ensuring agent explore the new actions.

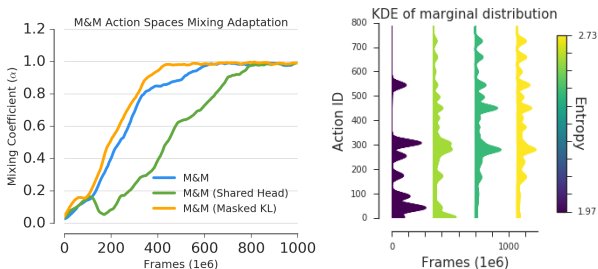


Figure 5. *Left*: Exemplary α value through time from a single run of an action space experiment. *Right*: Progression of marginal distribution of actions taken by the agent. Notice how the collision entropy ($-\mathbb{E}[\log \sum_a \pi_{mm}^2(a|s)]$) grows over time.

When plotting α through time (Fig. 5 *Left*) we see that the agent switches fully to the big action space early on, thus showing that small action space was useful only for initial phase of learning. This is further confirmed by looking

at how varied the actions taken by the agent are through training. Fig. 5 (*Right*) shows how the marginal distribution over actions evolves through time. We see that new actions are unlocked through training, and further that the final distribution is more entropic that the initial one.

4.2 Curricula over agent architecture

Another possible curriculum is over the main computational core of the agent. We use an architecture analogous to the one used in previous sections, but for the simple or initial agent, we substitute the LSTM with a linear projection from the processed convolutional signal onto a 256 dimensional latent space. We share both the convolutional modules as well as the policy/value function projections (Fig. 2b). We use a 540 element action space, and a factorised policy as described in the previous section.

We ran experiments on four problems in the DM Lab environment, focusing on various navigation tasks. On one hand, reactive policies (which can be represented solely by a FF policy) should learn reasonably quickly to move around and explore, while on the other hand, recurrent networks (which have memory) are needed to maximise the final performance – by either learning to navigate new maze layouts (Explore Object Location Small) or avoiding (seeking) explored unsuccessful (successful) paths through the maze.

As one can see on the average human normalised performance plot (Fig. 7) the M&M applied to the transition between FF and LSTM cores does lead to a significant improvement in final performance (20% increase in human normalised performance over tasks of interest). It is, however no longer as fast as the FF counterpart. In order to investigate this phenomenon we ran multiple ablation experiments (Fig. 6). In the first one, denoted FF+LSTM we use a skip connection which simply adds the activations of the FF core and LSTM core before passing it to a single linear projector for the policy/value heads. This enriched archi-

M&M- Agent Curricula for RL

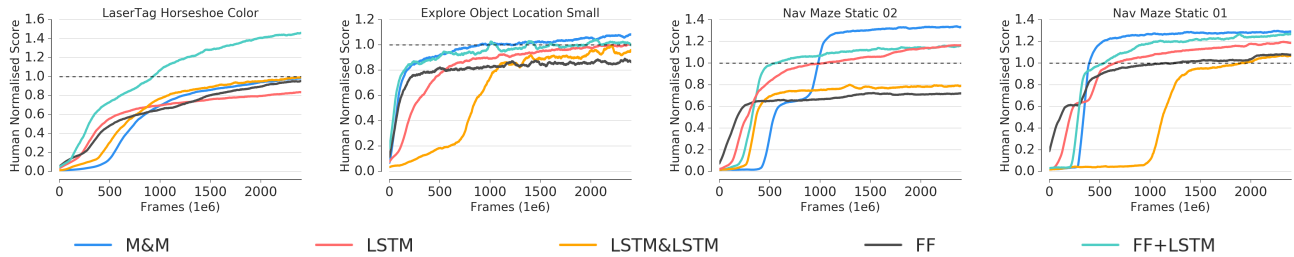


Figure 6. Comparison of the M&M agent and various baselines on four DM Lab levels. Each curve represents the average of 3 independent runs of 10 agents each (used for population based training). FF and LSTM represent the feedforward and LSTM baselines respectively, while FF+LSTM is a model with both cores and a skip connection. FF+LSTM is thus a significantly bigger model than the others, possibly explaining the outlier on the LT level. The LSTM&LSTM experiment shows M&M applied with two LSTM agents.

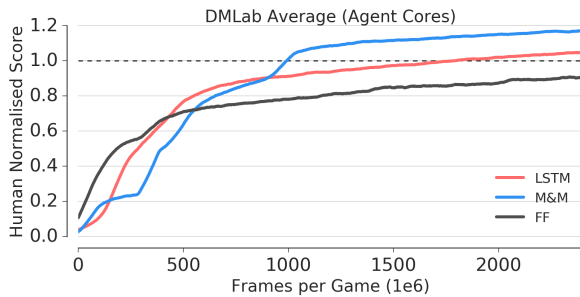


Figure 7. Human normalised score through training in **agent's core** experiments. Averaged across levels (for per level curves see Fig. 6).

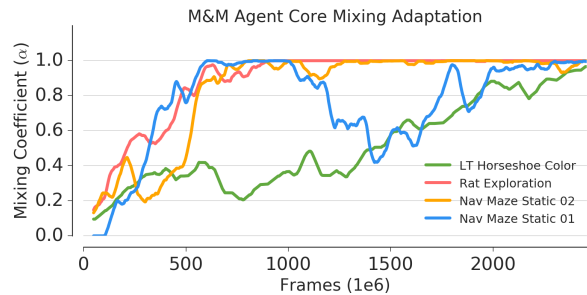


Figure 8. Exemplary α values through time from single runs of four DMLab levels. We find that our PBT optimisation finds curricula of different lengths as suited to the target problem.

ture does improve performance of LSTM only model, however it usually learns even slower, and has very similar learning dynamics to M&M. Consequently it strongly suggests that M&M's lack of initial speedup comes from the fact that it is architecturally more similar to the skip connection architecture. Note, that FF+LSTM is however a significantly bigger model (which appears to be helpful on LT Horseshoe color task).

Another question of interest is whether the benefit truly comes from the two core types, or simply through some sort of regularisation effect introduced by the KL cost. To test this hypothesis we also ran an M&M-like model but with 2 LSTM cores (instead of the feedforward). This model significantly underperforms all other baselines in speed and performance. This seems to suggest that the distillation or KL cost on its own is not responsible for any benefits we are seeing, and rather it is the full proposed Mix & Match method.

Finally if we look at the progression of the mixing coefficient (α) through time (Fig. 8), we notice once again quick switches on navigation-like tasks (all curves except the green one). However, there are two interesting observations to be made. First, the lasertag level, which requires a lot of reactivity in the policy, takes much longer to

switch to the LSTM core (however it does so eventually). This might be related to complexity of the level, which has pickup gadgets as well as many opponents, making memory useful much later in training. Secondly, for the simple goal finding task in a fixed maze (Nav maze static 01, the blue curve) the agent first rapidly switches to the LSTM, but then more or less mid training switches to the mixture policy ($\alpha \approx 0.5$) while finally switch completely again towards the end of training. This particular behaviour is possible due to the use of unconstrained α adaptation with PBT – thus depending on the current performance the agent can go back and forth through curriculum, which for this particular problem seems to be needed.

4.3 Curricula for multitask

As a final proof of concept we consider the task of learning a single policy capable of solving multiple RL problems at the same time. The basic approach for this sort of task is to train a model in a mixture of environments or equivalently to train a shared model in multiple environments in parallel (Teh et al., 2017; Espeholt et al., 2018). However, this sort of training can suffer from two drawbacks. First, it is heavily reward scale dependent, and will be biased towards high-reward environments. Second, environments that are

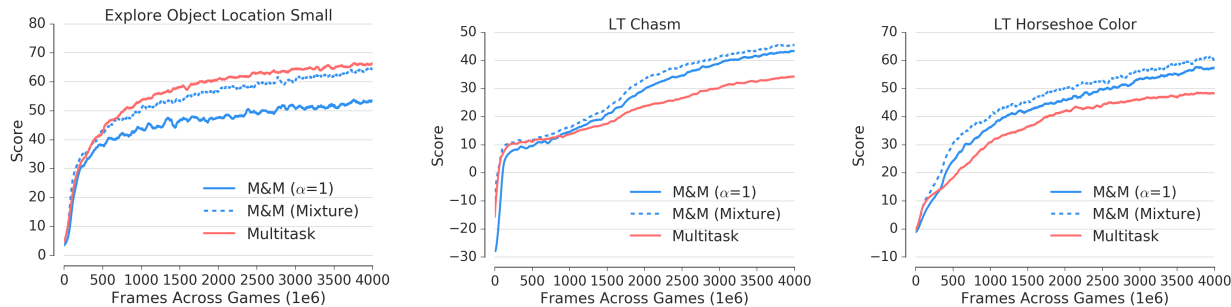


Figure 9. Performance of M&M applied to the multitask domain on three problems considered. Note that this is a performance of an agent trained jointly on all these tasks. The x-axis counts frames across tasks and the y-axis shows score per episode.

easy to train provide a lot of updates for the model and consequently can also bias the solution towards themselves.

To demonstrate this issue we use three DeepMind Lab environments – one is Explore Object Locations Small, which has high rewards and a steep initial learning curve (due to lots of reward signal coming from gathering apples). The two remaining ones are challenging laser tag levels (described in detail in the appendix). In both these problems training is hard, as the agent is interacting with other bots as well as complex mechanics (pick up bonuses, tagging floors, etc.).

We see in Fig. 9 that the multitask solution focuses on solving the navigation task, while performing comparatively poorly on the more challenging problems. To apply M&M to this problem we construct one agent per environment (each acting as π_1 from previous sections) and then one centralised “multitask” agent (π_2 from previous sections). Crucially, agents share convolutional layers but have independent LSTMs. Training is done in a multitask way, but the control policy in each environment is again a mixture between the task specific π_i (the specialist) and π_{mt} (centralised agent), see Fig. 2c for details. Since it is no longer beneficial to switch to the centralised policy, we use the performance of π_{mt} (i.e. the central policy) as the optimisation criterion (eval) for PBT, instead of the control policy.

We evaluate both the performance of the mixture and the centralised agent independently. Fig. 9 shows per task performance of the proposed method. One can notice much more uniform performance – the M&M agent learns to play well in both challenging laser tag environments, while slightly sacrificing performance in a single navigation task. One of the reasons of this success is the fact that knowledge transfer is done in policy space, which is invariant to reward scaling. While the agent can still focus purely on high reward environments once it has switched to using only the central policy, this inductive bias in training with M&M ensures a much higher minimum score.

5 Conclusions

We have demonstrated that the proposed method – Mix & Match – is an effective training framework to both improve final performance and accelerate the learning process for complex agents in challenging environments. This is achieved by constructing an implicit curriculum over agents of different training complexities. The collection of agents is bound together as a single composite whole using a mixture policy. Information can be shared between the components via shared experience or shared architectural elements, and also through a distillation-like KL-matching loss. Over time the component weightings of this mixture are adapted such that at the end of training we are left with a single active component consisting of the most complex agent – our main agent of interest from the outset. From an implementation perspective, the proposed method can be seen as a simple wrapper (the M&M wrapper) that is compatible with existing agent architectures and training schemes; as such it could easily be introduced as an additional element in conjunction with wide variety of on- or off-policy RL algorithms. In particular we note that, despite our focus on policy-based agents in this paper, the principles behind Mix & Match are also easily applied to value-based approaches such as Q-learning.

By leveraging M&M training, we are able to train complex agents much more effectively and in much less time than is possible if one were to attempt to train such an agent without the support of our methods. The diverse applications presented in this paper support the generality of our approach. We believe our training framework could help the community unlock the potential of powerful, but hitherto intractable, agent variants.

Acknowledgements

We would like to thank Raia Hadsell, Koray Kavukcuoglu, Lasse Espeholt and Iain Dunning for their invaluable comments, advice and support.

References

- Ba, J. and Caruana, R. Do deep nets really need to be deep? In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 27*, pp. 2654–2662. 2014.
- Beattie, C., Leibo, J. Z., Teplyaev, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., Legg, S., and Petersen, S. Deepmind lab. *CoRR*, 2016.
- Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *ICML*, 2009.
- Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
- Bucilu, C., Caruana, R., and Niculescu-Mizil, A. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541. ACM, 2006.
- Chen, T., Goodfellow, I. J., and Shlens, J. Net2net: Accelerating learning via knowledge transfer. *ICLR*, abs/1511.05641, 2016.
- Elman, J. Learning and development in neural networks: The importance of starting small. In *Cognition*, pp. 71–99, 1993.
- Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, 2018.
- Graves, A., Bellemare, M. G., Menick, J., Munos, R., and Kavukcuoglu, K. Automated curriculum learning for neural networks. *CoRR*, 2017.
- Heess, N., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, A., Riedmiller, M., et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.
- Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
- Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.
- Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C., and Kavukcuoglu, K. Population based training of neural networks. *CoRR*, 2017a.
- Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. Reinforcement learning with unsupervised auxiliary tasks. *ICLR*, 2017b.
- Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, pp. 1–8. IEEE, 2016.
- Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A. J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., et al. Learning to navigate in complex environments. *ICLR*, 2017.
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.
- Parisotto, E., Ba, L. J., and Salakhutdinov, R. Actor-mimic: Deep multitask and transfer reinforcement learning. *ICLR*, 2016.
- Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635, 2011.
- Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. Policy distillation. 2016.
- Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
- Sutskever, I. and Zaremba, W. Learning to execute. *CoRR*, 2014.
- Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. Distral: Robust multitask reinforcement learning. In *NIPS*. 2017.
- Wei, T., Wang, C., Rui, Y., and Chen, C. W. Network morphism. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 564–572, 2016.