CoBERL: Contrastive BERT for Reinforcement Learning

Anonymous Authors[1]

Abstract

Many reinforcement learning (RL) agents require a large amount of experience to solve tasks. We propose Contrastive BERT for RL (COBERL), an agent that combines a new contrastive loss and a hybrid LSTM-transformer architecture to tackle the challenge of improving data efficiency. COBERL enables efficient, robust learning from pixels across a wide range of domains. We use bidirectional masked prediction in combination with a generalization of recent contrastive methods to learn better representations for transformers in RL, without the need of hand engineered data augmentations. We find that COBERL consistently improves performance across the full Atari suite, a set of control tasks and a challenging 3D environment.

1. Introduction

Developing sample efficient reinforcement learning (RL) agents that only rely on raw high dimensional inputs is challenging. Specifically, it is difficult since it often requires to simultaneously train several neural networks based on sparse environment feedback and strong correlation between consecutive observation. This problem is particularly severe when the networks are large and densely connected, like in the case of the transformer (Vaswani et al., 2017) due to noisy gradients often found in RL problems. Outside of the RL domain, transformers have proven to be very expressive (Brown et al., 2020) and, as such, they are of particular interest for complex domains like RL.

In this paper, we propose to tackle this shortcoming by taking inspiration from Bidirectional Encoder Representations for Transformers (BERT; Devlin et al., 2019), and its successes on difficult sequence prediction and reasoning tasks. Specifically we propose a novel agent, named Contrastive BERT for RL (COBERL), that combines a new contrastive representation learning objective with architectural improvements that effectively combine LSTMs with transformers.

For representation learning we take inspiration from previous work showing that contrastive objectives improve the performance of agents (Fortunato et al., 2019; Srinivas et al., 2020b; Kostrikov et al., 2020; Mitrovic et al., 2021). Specifically, we combine the paradigm of masked prediction from BERT (Devlin et al., 2019) with the contrastive approach of Representation Learning via Invariant Causal Mechanisms (RELIC; Mitrovic et al., 2021). Extending the BERT masked prediction to RL is not trivial; unlike in language tasks, there are no discrete targets in RL. To circumvent this issue, we extend RELIC to the time domain and use it as a proxy supervision signal for the masked prediction. Such a signal aims to learn self-attention-consistent representations that contain the appropriate information for the agent to effectively incorporate previously observed knowledge in the transformer weights. Critically, this objective can be applied to different RL domains as it does not require any data augmentations and thus circumvents the need for much domain knowledge.

In terms of architecture, we base COBERL on Gated Transformer-XL (GTrXL; Parisotto et al., 2020) and Long-Short Term Memories (LSTMs; Hochreiter & Schmidhuber, 1997). GTrXL is an adaptation of a transformer architecture specific for RL domains. We combine GTrXL with LSTMs using a gate trained using RL gradients. This allows the agent to learn to exploit the representations offered by the transformer only when an environment requires it, and avoid the extra complexity when not needed.

We extensively test our proposed agent across a widely varied set of environments and tasks ranging from 2D platform games to 3D first-person and third-person view tasks. Specifically, we test it in the control domain using DeepMind Control Suite (Tassa et al., 2018) and probe its memory abilities using DMLab-30 (Beattie et al., 2016). We also test our agent on all 57 Atari games (Bellemare et al., 2013). Our main contributions are:

- A novel contrastive representation learning objective that combines the masked prediction from BERT with a generalization of RELIC to the time domain; with this we learn self-attention consistent representations and extend BERT-like training to RL and contrastive objectives, without the need of hand-engineered augmentations.
- An improved architecture that, using a gate, allows COBERL to flexibly combine transformer and an

LSTM.

- Improved performance on a varied set of environments and tasks both in terms of absolute performance and data efficiency. Also, we show that individually both our contrastive loss and the architecture improvements play a role in improving performance.

2. Method

To tackle the problem of data efficiency in deep reinforcement learning, we propose two modifications to the status quo. First, we introduce a novel representation learning objective aimed at learning better representations by enforcing self-attention consistency in the prediction of masked inputs. Second, we propose an architectural improvement to combine the strength of LSTMs and transformers.

2.1. Representation Learning

In RL the agent uses a batch of trajectories in order to optimize its RL objective. As part of this process, the observations in the trajectory are encoded into representations from which RL quantities of interest (e.g. value) are computed. Thus, learning informative representations is important for successfully solving the RL task at hand. Unlike in the supervised or unsupervised setting, learning representations in RL is complicated by (high) correlation between subsequent observations which we need to encode. Furthermore, given the often sparse reward signal coming from the environment learning representations in RL has to be achieved with little to no supervision.

To tackle these two issues, we propose to combine two approaches which have been successfully used in different domains, namely BERT (Devlin et al., 2019) and contrastive learning (Oord et al., 2018; Chen et al., 2020; Mitrovic et al., 2021). Here we borrow from BERT the combination of bidirectional processing in transformers (rather than left-to-right or right-to-left, as is common with RNN-based models such as LSTMs) with a masked prediction setup. The bidirectional processing allows the agent to learn the context of a particular state based on all of its temporal surroundings. On the other hand, predicting inputs at masked positions mitigates the issue of correlated inputs by reducing the probability of predicting subsequent time steps. Note that unlike in BERT where the input is a discrete vocabulary for language learning and we have targets available, in RL our inputs consist of images, rewards and actions that do not form a finite or discrete set and we are not given any targets. Thus, we must construct proxy targets and the corresponding proxy tasks to solve. For this we use contrastive learning. While many contrastive losses such as SimCLR (Chen et al., 2020) rely on data augmentations to create groupings of data that can be compared, we do not need to utilize these hand-crafted augmentations to construct proxy tasks. Instead we rely on the sequential nature of our input data to create the necessary groupings of similar and dissimilar points needed for contrastive learning and do not need to only rely on data augmentations (e.g. cropping, pixel variation) on image observations. As our contrastive loss, we use RELIC (Mitrovic et al., 2021) and adapt it to the time domain; we create the data groupings by aligning the input and output of the GTrXL transformer. We use RELIC as its KL regularization improves performance over approaches such as SimCLR (Chen et al., 2020) both in the domain of image classification as well as RL domains such as Atari as shown in (Mitrovic et al., 2021).

In a batch of sampled sequences, before feeding embeddings into the transformer stack, 15% of the embeddings are replaced with a fixed token denoting masking. Then, let the set $\mathcal{T}$ represent indices in the sequence that have been randomly masked and let $t \in \mathcal{T}$. For the i-th training sequence in the batch, for each index $t \in \mathcal{T}$, let x_t^i be the output of the GTrXL and y_t^i the corresponding input to the GTrXL from the encoder (see Fig 1B). Let $\phi(\cdot, \cdot)$ be the inner product defined on the space of critic embeddings, i.e. $\phi(x, y) = g(x)^T g(y)$, where g is a critic function. The critic provides separation between the embeddings used for the contrastive proxy task and the downstream RL task. Details of the critic function are in App. B. This separation is needed since the proxy and downstream tasks are related but not identical, and as such the appropriate representations will likely not be the same. As a side benefit, the critic can be used to reduce the dimensionality for the dot-product. To learn the embedding x_t^i at mask locations t, we use y_t^i as a positive example and the sets $\{y_t^b\}_{b=0,b\neq i}^B$ and $\{x_t^b\}_{b=0,b\neq i}^B$ as the negative examples with B the number of sequences in the minibatch. We model q_x^t as

$$q_x^t = \frac{\exp(\phi(x_t^i, y_t^i))}{\sum_{b=0}^B \exp(\phi(x_t^i, y_t^b)) + \exp(\phi(x_t^i, x_t^b))} \tag{1}$$

with q_x^t denoting $q_x^t\left(x_t^i | \{y_t^b\}_{b=0}^B, \{x_t^b\}_{b=0,b\neq i}^B\right)$; q_y^t is computed analogously (see Fig 1C). In order to enforce self-attention consistency in the learned representations, we explicitly regularize the similarity between the pairs of transformer embeddings and inputs through Kullback-Leibler regularization from RELIC. Specifically, we look at the similarity between appropriate embeddings and inputs, and within the sets of embeddings and inputs separately. To this end, we define

$$p_x^t = \frac{\exp(\phi(x_t^i, y_t^i))}{\sum_{b=0}^B \exp(\phi(x_t^i, y_t^b))} \tag{2}$$

and

$$s_x^t = \frac{\exp(\phi(x_t^i, x_t^i))}{\sum_{b=0}^B \exp(\phi(x_t^i, x_t^b))} \tag{3}$$

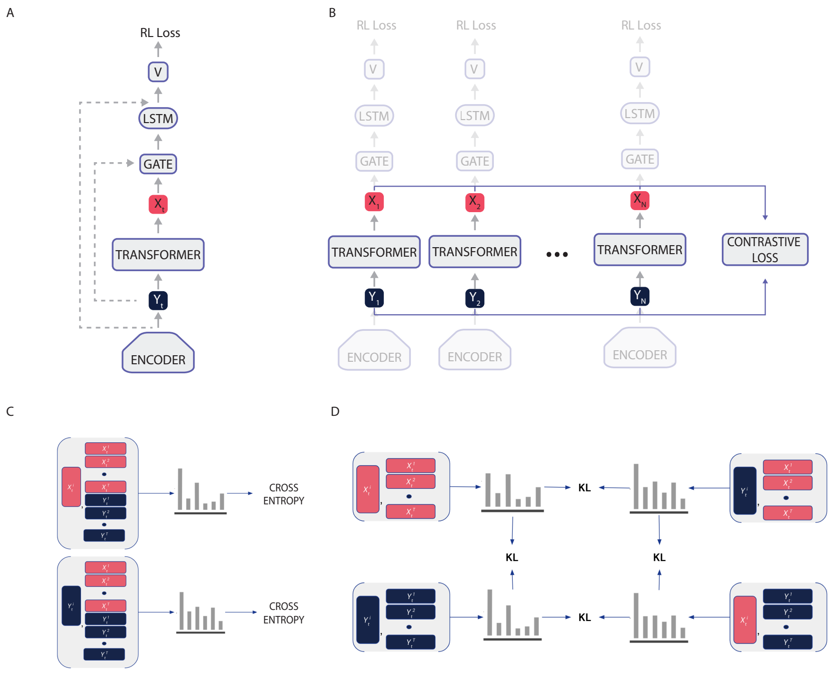

Figure 1. CoBERL. A) General Architecture. We use a residual network to encode observations into embeddings Y_t. We feed Y_t through a causally masked GTrXL transformer, which computes the predicted masked inputs X_t and passes those together with Y_t to a learnt gate. The output of the gate is passed through a single LSTM layer to produce the values that we use for computing the RL loss. B) Contrastive loss. We also compute a contrastive loss using predicted masked inputs X_t and Y_t as targets. For this, we do not use the causal mask of the Transfomer. For details about the contrastive loss, please see Section 2.1. C) Computation of q_x^t and q_y^t (from Equation 1) with the round brackets denoting the computation of the similarity between the entries D) Regularization terms from Eq. 4 which explicitly enforce self-attention consistency.

with p_x^t and s_x^t shorthand for $p_x^t(x_t^i|\{y_t^b\}_{b=0}^B)$ and $s_x^t(x_t^i|\{x_t^b\}_{b=0}^B)$, respectively; $p_y^t(y_t^i|\{x_t^b\}_{b=0}^B)$ and $s_y^t(y_t^i|\{y_t^b\}_{b=0}^B)$ defined analogously (see Fig 1D). Putting together the individual contributions, the final objective takes the form

$$\mathcal{L}(X,Y) = -\sum_{t\in\mathcal{T}}(\log q_x^t + \log q_y^t) \\ +\alpha\sum_{t\in\mathcal{T}}\Big[KL(s_x^t, \mathrm{sg}(s_y^t)) + KL(p_x^t, \mathrm{sg}(p_y^t)) \\ +KL(p_x^t, \mathrm{sg}(s_y^t)) + KL(p_y^t, \mathrm{sg}(s_x^t))\Big] \tag{4}$$

with $\mathrm{sg}(\cdot)$ indicating a stop-gradient.

Identically to our RL objective, we use the full batch of sequences that are sampled from the replay buffer to optimize this contrastive objective. In practice, we optimize a weighted sum of the RL objective and $\mathcal{L}(X,Y)$.

2.2. Architecture of CoBERL.

While transformers have proven very effective at connecting long-range data dependencies in natural language processing (Vaswani et al., 2017; Brown et al., 2020; Devlin et al., 2019) and computer vision (Carion et al., 2020; Dosovitskiy et al., 2021), in the RL setting they are difficult to train and are prone to overfitting (Parisotto et al., 2020). In contrast, LSTMs have long been demonstrated to be useful in RL. Although less able to capture long range dependencies due to their sequential nature, LSTMs capture recent dependencies effectively. We propose a simple but powerful architectural change: we add an LSTM layer on top of the GTrXL with an extra gated residual connection between the LSTM and GTrXL, modulated by the input to the GTrXL (see Fig 1A). Finally we also have a skip connection from the transformer input to the LSTM output.

More concretely, let Y_t be the output of the encoder network at time t, then the additional module can be defined by the following equations (see Fig 1A, Gate(), below, has the same form as other gates internal to GTrXL):

$$X_t = \mathrm{GTrXL}(Y_t) \tag{5}$$
$$Z_t = \mathrm{Gate}(Y_t, X_t) \tag{6}$$
$$\mathrm{Output}_t = \mathrm{concatenate}(\mathrm{LSTM}(Z_t), Y_t) \tag{7}$$

These modules are complementary as the transformer has no recency bias (Ravfogel et al., 2019), whilst the LSTM is biased to represent more recent inputs - the gate in equation 6 allows this to be a mix of encoder representations and transformer outputs. This memory architecture is agnostic to the choice of RL regime and we evaluate this architecture in both the on and off-policy settings. For on-policy, we use V-MPO(Song et al., 2019) as our RL algorithm. V-MPO uses a target distribution for policy updates, and partially moves the parameters towards this target subject to KL constraints. For the off-policy setting, we use R2D2 (Kapturowski et al., 2018) which adapts replay and the RL learning objective for agents with recurrent architectures, such as LSTMs, GTrXL, and CoBERL.

R2D2 Agent Recurrent Replay Distributed DQN (R2D2; Kapturowski et al., 2018) demonstrates how replay and the RL learning objective can be adapted to work well for agents with recurrent architectures. Given its competitive performance on Atari-57 and DMLab-30, we implement our CoBERL architecture in the context of Recurrent Replay Distributed DQN (Kapturowski et al., 2018). We effectively replace the LSTM with our gated transformer and LSTM combination and add the contrastive representation learning loss. With R2D2 we thus leverage the benefits of distributed experience collection, storing the recurrent agent state in the replay buffer, and "burning in" a portion of the unrolled network with replayed sequences during training.

V-MPO Agent Given V-MPO's strong performance on DMLab-30, in particular in conjunction with the GTrXL architecture (Parisotto et al., 2020) which is a key component of CoBERL, we use V-MPO and DMLab-30 to demonstrate CoBERL's use with on-policy algorithms. V-MPO is an on-policy adaptation of Maximum a Posteriori Policy Optimization (MPO) (Abdolmaleki et al., 2018). To avoid high variance often found in policy gradient methods, V-MPO uses a target distribution for policy updates, subject to a sample-based KL constraint, and gradients are calculated to partially move the parameters towards the target, again subject to a KL constraint. Unlike MPO, V-MPO uses a learned state-value function $V(s)$ instead of a state-action value function.

3. Related Work

Transformers in RL. The transformer architecture (Vaswani et al., 2017) has recently emerged as one of the best performing approaches in language modelling (Dai et al., 2019; Brown et al., 2020) and question answering (Dehghani et al., 2018; Yang et al., 2019). More recently it has also been successfully applied to computer vision (Dosovitskiy et al., 2021). Given the similarities of sequential data processing in language modelling and reinforcement learning, transformers have also been successfully applied to the RL domain, where as motivation for GTrXL, (Parisotto et al., 2020) noted that extra gating was helpful to train transformers for RL due to the high variance of the gradients in RL relative to that of (un)supervised learning problems. In this work, we build upon GTrXL and demonstrate that, perhaps for RL: attention is not all you need, and by combining GTrXL in the right way with an LSTM, superior performance is attained. We reason that this demonstrates the advantage of both forms of memory representation: the all-to-all attention of transformers combined with the sequential processing of LSTMs. In doing so, we demonstrate that care should be taken in how LSTMs and transformers are combined and show a simple gating is most effective in our experiments. Also, unlike GTrXL, we show that using an unsupervised representation learning loss that enforces self-attention consistency is a way to enhance data efficiency when using transformers in RL.

Contrastive Learning. Recently contrastive learning (Hadsell et al., 2006; Gutmann & Hyvärinen, 2010; Oord et al., 2018) has emerged as a very performant paradigm for unsupervised representation learning, in some cases even surpassing supervised learning (Chen et al., 2020; Caron et al., 2020; Mitrovic et al., 2021). These methods have also been leveraged in an RL setting with the hope of improving performance. Apart from MRA (Fortunato et al., 2019) mentioned above, one of the early examples of this is CURL (Srinivas et al., 2020a) which combines Q-Learning with a separate encoder used for representation learning with the InfoNCE loss from CPC (Oord et al., 2018). More recent examples use contrastive learning for predicting future latent states (Schwarzer et al., 2020; Mazoure et al., 2020), defining a policy similarity embeddings (Agarwal et al., 2021) and learning abstract representations of state-action pairs (Liu et al., 2021). The closest work to our is M-CURL (Zhu et al., 2020). Like our work, it combines mask prediction, transformers, and contrastive learning, but there are a few key differences. First, unlike M-CURL who use a separate policy network, CoBERL computes Q-values based on the output of the transformer. Second, CoBERL combines the transformer architecture with GRUs to produce the input for the Q-network, while M-CURL uses the transformer as an additional embedding network (critic) for the computation of the contrastive loss. Third, while CoBERL uses an extension of ReLIC (Mitrovic et al., 2021) to the time domain and operates on the inputs and outputs of the transformer, M-CURL uses CPC (Oord et al., 2018) with a momentum encoder as in (Srinivas et al., 2020a) and compares encodings from the transformer with the separate momentum encoder.

4. Experiments

We provide empirical evidence to show that CoBERL i) improves performance across a wide range of environments and tasks, and ii) needs all its components to maximise its performance. In our experiments, we demonstrate performance on Atari57 (Bellemare et al., 2013), the DeepMind Control Suite (Tassa et al., 2018), and the DMLab-30 (Beattie et al., 2016). Recently, Dreamer V2 (Hafner et al., 2020b) has emerged as a strong model-based agent across Atari57 and DeepMind Control Suite; we therefore include it as a reference point for performance on these domains. All results are averaged over three seeds, with standard error reported on mean performance (see App. A.1 for details).

For the experiments in Atari57 and the DeepMind Control suite, CoBERL uses the R2D2 distributed setup. We use 512 actors for all our experiments. We do not constrain the amount of replay done for each experience trajectory that actors deposit in the buffer. However, we have found empirical replay frequency per data point to be close among all our experiments (with an expected value of 1.5 samples per data point). We use a separate evaluator process that shares weights with our learner in order to measure the performance of our agents. We report scores at the end of training. A more comprehensive description of the setup of this distributed system is found in App. A.1. CoBERL and the baselines that we used in Atari57 and the DeepMind Control suite employ the same 47-layer ResNet to encode observations. Details on the parts and size of all the components of the architecture of CoBERL, including the sizes used for the transformer and LSTM parts, are described in App. B. The hyperparameters and architecture we choose for these two domains are the same with two exceptions: i) we use a shorter trace length for Atari (80 instead of 120) as the environment does not require a long context to inform decisions, and ii) we use a squashing function on Atari and the Control Suite to transform our Q values (as done in (Kapturowski et al., 2018)) since reward structures vary highly in magnitude between tasks. We use Peng's $Q(\lambda)$ as our loss. To ensure that this is comparable to R2D2, we also run an R2D2 baseline with this loss. All results are shown as the average over 3 seeds. A comprehensive enumeration of the hyperparameters we use are shown in App. C.

For DMLab-30 we use V-MPO(Song et al., 2019) to directly compare CoBERL with (Parisotto et al., 2020) and also demonstrate how CoBERL may be applied to both on and off-policy learning. The experiments were run using a Podracer setup (Hessel et al., 2021), details of which may be found in App. A.2. CoBERL is trained for 10 billion steps on all 30 DMLab-30 games at the same time, to mirror the exact multi-task setup presented in (Parisotto et al., 2020). Compared to (Parisotto et al., 2020) we have two differences. Firstly, all the networks run without pixel control loss (Jaderberg et al., 2016) so as not to confound our contrastive loss with the pixel control loss. Secondly all the models used a fixed set of hyperparameters with 3 random seeds, whereas in (Parisotto et al., 2020) the results were averaged across hyperparameters. Here the hyperparameters were chosen to maximise the performance of the GTrXL baseline, please see App. C for more details.

4.1. CoBERL as a General Agent

To test the generality of our approach, we analyze the performance of our model on a wide range of environments. We show results on the Arcade Learning Environment (Bellemare et al., 2013), DeepMind Lab (Beattie et al., 2016), as well as the DeepMind Control Suite (Tassa et al., 2018). To help with comparisons, in Atari-57 and DeepMind Control we introduce an additional baseline, which we name R2D2-GTrXL. This baseline is a variant of R2D2 where the LSTM is replaced by GTrXL. R2D2-GTrXL has no unsupervised learning. This way we are able to observe how GTrXL is affected by the change to an off-policy agent (R2D2), from its original V-MPO implementation in (Parisotto et al., 2020). We also perform an additional ablation analysis by removing the contrastive loss from CoBERL (see Sec. 4.2.1). With this baseline we demonstrate the importance of contrastive learning in these domains, and we show that the combination of an LSTM and transformer is superior to either alone.

Atari As commonly done in literature (Mnih et al., 2015; Hessel et al., 2018; Machado et al., 2018; Hafner et al., 2020b), we measure performance on all 57 Atari games after running for 200 million frames. As detailed in App. C, we use the standard Atari frame pre-processing to obtain the 84x84 gray-scaled frames that are used as input to our agent. We do not use frame stacking.

	$>$ h.	Mean	Median	25th Pct	5th Pct
CoBERL	**49**	**1424.9% ± 43.30%**	276.6%	**149.3%**	**17.0%**
R2D2-GTrXL	48	1201.6% ± 16.63%	**313.7%**	139.6%	3.7%
R2D2	47	1024.2% ± 40.11%	272.6%	138.1%	3.3%
Rainbow	43	874.0%	231.0%	101.7%	4.9%
Dreamer V2*	37	631.1%	162.0%	76.6%	2.5%

Table 1. The human normalized scores on Atari-57. $>$ h indicates the number of tasks for which performance above average human was achieved. $*$ indicates that it was run on 55 games with sticky actions; Pct refers to percentile.

Tab. 1 shows the results of all the agents where published results are available. CoBERL shows the most games above average human performance and significantly higher overall mean performance. Interestingly, the performance of R2D2-GTrXL shows that the addition of GTrXL is not sufficient to obtain the improvement in performance that CoBERL exhibits–below we will demonstrate through ablations that both the contrastive loss and LSTM contribute to this improvement. R2D2-GTrXL also exhibits slightly better median than CoBERL, showing that R2D2-GTrXL is indeed a powerful variant on Atari. Additionally, we observe that the difference in performance in CoBERL is higher when examining the lower percentiles. This suggests that CoBERL causes an improvement in data efficiency, since, as shown in experiments in (Kapturowski et al., 2018) which are run for billions of frames, these results are far from the final performance of R2D2.

Control Suite We also perform experiments on the DeepMind Control Suite (Tassa et al., 2018). While the action space in this domain is typically treated as continuous, we discretize the action space in our experiments and apply the same architecture as in Atari and DMLab-30. For more details on the number of actions for each task see App. C.4. We do not use pre-processing on the frames received from the environment. Finally, CoBERL is trained only from pixels without state information.

We include six tasks popular in current literature: `ball_in_cup catch`, `cartpole swingup`, `cheetah run`, `finger spin`, `reacher easy`, and `walker walk`. Most previous work on these specific tasks has emphasized data efficiency as most are trivial to solve even with the baseline—D4PG-Pixels—in the original dataset paper (Tassa et al., 2018). We thus include 6 other tasks that are difficult to solve with D4PG-Pixels and are relatively less explored: `acrobot swingup`, `cartpole swingup_sparse`, `fish swim`, `fish upright`, `pendulum swingup`, and `swimmer swimmer6`. We show our results in Table 2. We show results on CoBERL, R2D2-gTRXL, R2D2, CURL (Srinivas et al., 2020a), Dreamer (Hafner et al., 2020a), Soft Actor Critic (Haarnoja et al., 2018) on pixels as demonstrated in (Srinivas et al., 2020a), and D4PG-Pixels (Tassa et al., 2018). CURL, DREAMER, and Pixel SAC are for reference only as they represent the state the art for low-data experiments (500K environment steps). These three are not perfectly comparable baselines; however, D4PG-Pixels is run on a comparable scale with 100 million environment steps. Because CoBERL relies on large scale distributed experience, we have a much larger number of available environment steps per gradient update. We run for 100M environment steps as with D4PG-Pixels, and we compute performance for our approaches by taking the evaluation performance of the final 10% of steps. Across the majority of tasks, CoBERL outperforms D4PG-Pixels. The increase in performance is especially apparent for the more difficult tasks. For most of the easier tasks, the performance difference between the CoBERL, R2D2-GTrXL, and R2D2 is negligible. For `ball_in_cup catch`, `cartpole swingup`, `finger spin` and `reacher easy`, even the original R2D2 agent performs on par with the D4PG-Pixels baseline. On more difficult tasks such as `fish swim`, and `swimmer swimmer6`, there is a very large, appreciable difference between CoBERL, R2D2, and R2D2-GTrXL. The combination of the LSTM and transformer specifically makes a large difference here especially compared to D4PG-Pixels. Interestingly, this architecture is also very important for situations where the R2D2-based approaches underperform. For `cheetah run` and `walker walk`, the CoBERL architecture dramatically narrows the performance gap between the R2D2 agent and the state of the art.

DMLab-30. To test CoBERL in a challenging 3 dimensional environment we run it in DmLab-30 (Beattie et al., 2016). The agent was trained at the same time on all the 30 tasks, following the setup of GTrXl (Parisotto et al., 2020), which we use as our baseline. In Figure 2A we show the final results on the DMLab-30 domain. If we look at all the 30 games, CoBERL reaches a substantially higher score than GTrXL (CoBERL=113.39% $\pm$ 3.64%, GTrXL=102.40% $\pm$ 0.23%, Figure 2A). We also analysed the number of steps required to reach 100% human normalised score, a measure for data efficiency. In this respect, CoBERL requires considerably fewer environment frames than GTrXL (CoBERL=2.96 $\pm$ 0.35 Billion, GTrXL=3.64 $\pm$ 0.43 Billion, see Figure 2B).

4.2. Ablations

In Sec. 1, we explained contributions that are essential to CoBERL, the new contrastive learning loss, and the architectural changes. We now explore the effects of these two separate contributions, disentangling the added benefit of each separately. Moreover, we run a set of ablations to understand the role of model size on the results. Ablations are run on 7 Atari games chosen to match the ones in the original DQN publication (Mnih et al., 2013), and on all the 30 DMLab games.

4.2.1. Impact of auxiliary losses

In Table 3 we show that our contrastive loss contributes to a significant gain in performance, both in Atari and DMLab-30, when compared to CoBERL without it. Also, in challenging environments like DmLab-30, CoBERL without extra loss is still superior to the relative baseline. The only case where we do not see and advantage of using the auxiliary loss is if we consider the median score on the reduced ablation set of Atari games. However in the case of the DmLab-30, where we consider a larger set of levels (7 vs. 30), there is a clear benefit of the auxiliary loss.

Moreover, Table 4 reports a comparison between our loss, SimCLR (Chen et al., 2020) and CURL (Srinivas et al.,

DM Suite	CoBERL	R2D2-GTrXL	R2D2	D4PG-Pixels	CURL	Dreamer	Pixel SAC
acrobot swingup	**359.75 ± 3.47**	215.39 ± 122.82	327.16 ± 5.35	81.7 ± 4.4	-	-	-
fish swim	**624.40 ± 54.91**	91.32 ± 277.15	345.63 ± 227.44	72.2 ± 3.0	-	-	-
fish upright	**942.33 ± 6.12**	849.52 ± 23.01	936.09 ± 11.58	405.7 ± 19.6	-	-	-
pendulum swingup	**836.63 ± 9.77**	743.65 ± 52.44	831.86 ± 61.54	680.9 ± 41.9	-	-	-
swimmer swimmer6	**447.60 ± 51.51**	225.97 ± 60.67	329.61 ± 26.77	194.7 ± 15.9	-	-	-
finger spin	**985.05 ± 1.58**	977.41 ± 8.91	980.85 ± 0.67	**985.7 ± 0.6**	926 ± 45	796 ± 183	179 ± 166
reacher easy	**983.05 ± 2.47**	981.64 ± 1.99	**982.28 ± 9.30**	967.4 ± 4.1	929 ± 44	793 ± 164	145 ± 30
cheetah run	525.06 ± 44.59	115.15 ± 133.95	365.45 ± 50.40	523.8 ± 6.8	518 ± 28	**570 ± 253**	197 ± 15
walker walk	780.54 ± 26.48	595.96 ± 77.59	687.18 ± 18.15	**968.3 ± 1.8**	902 ± 43	897 ± 49	42 ± 12
ball in cup catch	978.28 ± 6.56	975.21 ± 1.77	**980.54 ± 1.94**	980.5 ± 0.5	959 ± 27	879 ± 87	312 ± 63
cartpole swingup	798.66 ± 7.72	837.31 ± 4.15	816.23 ± 2.93	**862.0 ± 1.1**	841 ± 45	762 ± 27	419 ± 40
cartpole swingup sparse	732.51 ± 18.60	747.94 ± 8.61	**762.57 ± 6.71**	482.0 ± 56.6	-	-	-

Table 2. Results on tasks in the DeepMind Control Suite. CoBERL, R2D2-GTrXL, R2D2, and D4PG-Pixels are trained on 100M frames, while CURL, Dreamer, and Pixel SAC are trained on 500k frames. We show these three other approaches as reference and not as a directly comparable baseline.

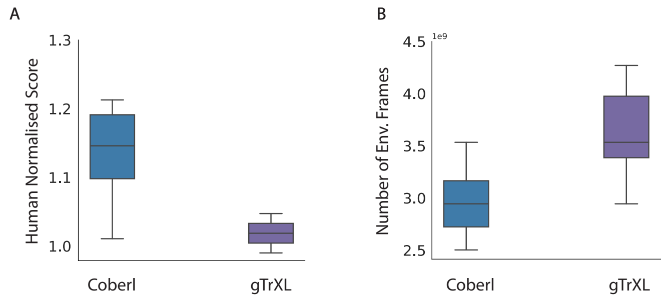

Figure 2. DMLab-30 experiments. a) Human normalised returns in DMLab-30 across all the 30 levels. b) Average number of steps to reach 100% human normalised score across all the 30 levels. Results are over 3 seeds and the final 5% of training.

2020a). Although simpler than both SimCLR - which in its original implementation requires handcrafted augmentations - and CURL - which requires an additional network - our contrastive method shows improved performance. These experiments where run only on Atari to reduce computational costs while still being sufficient for the analysis.

4.2.2. Impact of architectural changes

Table 5 shows the effects of removing the LSTM from CoBERL (column "w/o LSTM"), as well as removing the gate and its associated skip connection (column "w/o Gate"). In both cases CoBERL performs substantially worse showing that both components are needed. Finally, we also experimented with substituting the learned gate with either a sum or a concatenation. The results, presented in Appendix D, show that in most occasions these alternatives decrease performance, but not as substantially as removing the LSTM, gate or skip connections. Our hypothesis is that the learned gate should give more flexibility in complex environments, we leave it open for future work to do more analysis on this.

4.2.3. Impact of number of parameters

Table 6 compares the models in terms of the number of parameters. For Atari, the number of parameters added by CoBERL over the R2D2(GTrXL) baseline is very limited; however, CoBERL still produces a significant gain in performance. We also tried to move the LSTM before the transformer module (column "CoBERL with LSTM before"). In this case the representations for the contrastive loss were taken from before the LSTM. Interestingly, this setting performs worse, despite having the same number of parameters as CoBERL. For DMLab-30, it is worth noting that CoBERL has a memory size of 256, whereas GTrXL has a memory of size 512 resulting in substantially fewer parameters. Nevertheless, the discrepancies between models are even more pronounced, even though the number of parameters is either exactly the same ("CoBERL with LSTM before") or higher (GTrXL). This ablation is of particular interest as it shows that the results are driven by the particular architectural choice rather than the added parameters.

5. Limitations and Future Work

A limitation of our method is that it relies on single time step information to compute its auxiliary objective. Such objective could naturally be adapted to operate on temporally-extended patches, and/or action-conditioned inputs. We regard those ideas as promising future research avenues.

6. Conclusions

We proposed a novel RL agent, Contrastive BERT for RL (CoBERL), which introduces a new contrastive representation learning loss that enables the agent to efficiently learn

		CoBERL	CoBERL w/o aux loss	GTrXL baseline*
DMLab-30	Mean	$\mathbf{113.39\% \pm 3.64\%}$	$106.95\% \pm 1.41\%$	$102.40\% \pm 0.23\%$
	Median	$\mathbf{112.02\%}$	106.96%	103.40%
Atari	Mean	$\mathbf{698.01\% \pm 53.84\%}$	$578.41\% \pm 81.56\%$	$636.59\% \pm 44.43\%$
	Median	276.63%	$\mathbf{377.68\%}$	368.99%

Table 3. Impact of contrastive loss. Human normalized scores on Atari-57 ablation tasks and DMLab-30 tasks. * for DMLab the baseline is GTrXL trained with VMPO, for Atari the baseline is GTrXL trained with R2D2.

		CoBERL	CoBERL with CURL	CoBERL with SimCLR
Atari	Mean	$\mathbf{698.01\% \pm 53.84\%}$	$507.30\% \pm 72.19\%$	$635.76\% \pm 100.45\%$
	Median	$\mathbf{276.63\%}$	270.97%	272.46%

Table 4. Comparison with alternative auxiliary losses.

		CoBERL	w/o LSTM	w/o Gate
DMLab-30	Mean	$\mathbf{113.39\% \pm 3.64\%}$	$98.72\% \pm 5.50$	$84.07\% \pm 5.71\%$
	Median	$\mathbf{112.02\%}$	100.35%	104.33%
Atari	Mean	$\mathbf{698.0\% \pm 53.84}$	$433.92\% \pm 13.67\%$	$591.33\% \pm 91.25\%$
	Median	276.6%	259.77%	$\mathbf{320.09\%}$

Table 5. Impact of Architectural changes. Human normalized scores on Atari-57 ablation tasks and DMLab-30 tasks. * for DMLab-30 the baseline is GTrXL trained with VMPO, for Atari the baseline is GTrXL trained with R2D2.

		CoBERL	GTrXL*	CoBERL with LSTM before	R2D2 LSTM
DMLab-30	Mean	$\mathbf{113.39\% \pm 3.64\%}$	$102.40\% \pm 5.50\%$	$100.65\% \pm 1.80\%$	N/A
	Median	$\mathbf{112.02\%}$	103.40%	101.68%	N/A
	Num. Params.	47M	66M	47M	N/A
Atari	Mean	$\mathbf{698.0\% \pm 53.84}$	$636.59\% \pm 44.43\%$	$508.28\% \pm 70.20\%$	$353.99\% \pm 26.19\%$
	Median	$\mathbf{276.6\%}$	259.77%	271.50%	260.40%
	Num. Params.	46M	42M	46M	18M

Table 6. Effect of number of parameters. Human normalized scores on Atari-57 ablation tasks and DMLab-30 tasks. *for DMLab-30 the baseline is GTrXL trained with VMPO with a memory size of 512, for Atari the baseline is GTrXL trained with R2D2 with a memory size of 64.

consistent representations. This, paired with an improved architecture, resulted in better data efficiency and final scores on a varied set of environments and tasks. On Atari, CoBERL comfortably outperformed competing methods surpassing the human benchmark in 49 out of the 57 games. In the the DeepMind Control Suite, CoBERL showed significant improvement over previous work in 3 out of 6 tasks, while matching previous state-of-the-art in the remaining 3 tasks. On DMLab-30, CoBERL was significantly more data-efficient than GTrXl and also obtained higher final scores. Moreover, through an extensive set of ablation experiments we confirmed that all CoBERL components are necessary to achieve the performance of the final agent. To conclude, we have shown that our auxiliary loss and architecture provide an effective and general means to efficiently train large attentional models in RL.

References

Abdolmaleki, A., Springenberg, J. T., Degrave, J., Bohez, S., Tassa, Y., Belov, D., Heess, N., and Riedmiller, M. Relative entropy regularized policy iteration. *arXiv preprint arXiv:1812.02256*, 2018.

Agarwal, R., Machado, M. C., Castro, P. S., and Bellemare, M. G. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. *ArXiv*, abs/2101.05265, 2021.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M.

The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In *NeurIPS*, 2020.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. End-to-end object detection with transformers. In *ECCV*, 2020.

Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., and Joulin, A. Unsupervised learning of visual features by contrasting cluster assignments. *ArXiv*, abs/2006.09882, 2020.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 1597–1607, 2020.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, Ł. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2019.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Fortunato, M., Tan, M., Faulkner, R., Hansen, S., Badia, A. P., Buttimore, G., Deck, C., Leibo, J. Z., and Blundell, C. Generalization of reinforcement learners with working and episodic memory. *NeurIPS*, 2019.

Gutmann, M. and Hyvärinen, A. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 297–304, 2010.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

Hadsell, R., Chopra, S., and LeCun, Y. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pp. 1735–1742. IEEE, 2006.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. In *ICLR*, 2020a.

Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020b.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Hessel, M., Kroiss, M., Clark, A., Kemaev, I., Quan, J., Keck, T., Viola, F., and Hasselt, H. v. Podracer architectures for scalable reinforcement learning, 2021.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

Kostrikov, I., Yarats, D., and Fergus, R. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

Liu, G., Zhang, C., Zhao, L., Qin, T., Zhu, J., Jian, L., Yu, N., and Liu, T.-Y. Return-based contrastive representation learning for reinforcement learning. In *ICLR 2021*, January 2021.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Mazoure, B., des Combes, R. T., Doan, T., Bachman, P., and Hjelm, R. D. Deep reinforcement and infomax learning. *ArXiv*, abs/2006.07217, 2020.

Mitrovic, J., McWilliams, B., Walker, J., Buesing, L., and Blundell, C. Representation learning via invariant causal mechanisms. In *International conference on learning representations*, 2021.

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

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Oord, A. v. d., Li, Y., and Vinyals, O. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Parisotto, E., Song, F., Rae, J., Pascanu, R., Gulcehre, C., Jayakumar, S., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., et al. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning*, pp. 7487–7498. PMLR, 2020.

Ravfogel, S., Goldberg, Y., and Linzen, T. Studying the inductive biases of rnns with synthetic variations of natural languages. *arXiv preprint arXiv:1903.06400*, 2019.

Schwarzer, M., Anand, A., Goel, R., Hjelm, R. D., Courville, A. C., and Bachman, P. Data-efficient reinforcement learning with momentum predictive representations. *ArXiv*, abs/2007.05929, 2020.

Song, H. F., Abdolmaleki, A., Springenberg, J. T., Clark, A., Soyer, H., Rae, J. W., Noury, S., Ahuja, A., Liu, S., Tirumala, D., et al. V-mpo: On-policy maximum a posteriori policy optimization for discrete and continuous control. *arXiv preprint arXiv:1909.12238*, 2019.

Srinivas, A., Laskin, M., and Abbeel, P. Curl: Contrastive unsupervised representations for reinforcement learning. In *ICML*, 2020a.

Srinivas, A., Laskin, M., and Abbeel, P. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*, 2020b.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

van Hasselt, H., Guez, A., Hessel, M., Mnih, V., and Silver, D. Learning values across many orders of magnitude. *arXiv preprint arXiv:1602.07714*, 2016.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *NeurIPS*, 2017.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

Zhu, J., Xia, Y., Wu, L., Deng, J., Zhou, W., Qin, T., and Li, H. Masked contrastive representation learning for reinforcement learning. *ArXiv*, abs/2010.07470, 2020.

A. Setup details

A.1. R2D2 Distributed system setup

Following R2D2, the distributed system consists of several parts: actors, a replay buffer, a learner, and an evaluator. Additionally, we introduce a centralized batched inference process to make more efficient use of actor resources.

Actors: We use 512 processes to interact with independent copies of the environment, called actors. They send the following information to a central batch inference process:

- x_t: the observation at time t.
- r_{t-1}: the reward at the previous time, initialized with $r_{-1} = 0$.
- a_{t-1}: the action at the previous time, a_{-1} is initialized to 0.
- h_{t-1}: recurrent state at the previous time, is initialized with $h_{-1} = 0$.

They block until they receive $Q(x_t, a; \theta)$. The l-th actor picks a_t using an ϵ_l-greedy policy. As R2D2, the value of ϵ_l is computed following:

$$\epsilon_l = \epsilon^{1+\alpha \frac{l}{L-1}}$$

where $\epsilon = 0.4$ and $\alpha = 8$. After that is computed, the actors send the experienced transition information to the replay buffer.

Batch inference process: This central batch inference process receives the inputs mentioned above from all actors. This process has the same architecture as the learner with weights that are fetched from the learner every 0.5 seconds. The process blocks until a sufficient amount of actors have sent inputs, forming a batch. We use a batch size of 64 in our experiments. After a batch is formed, the neural network of the agent is run to compute $Q(x_t, a, \theta)$ for the whole batch, and these values are sent to their corresponding actors.

Replay buffer: it stores fixed-length sequences of *transitions* $T = (\omega_s)_{s=t}^{t+L-1}$ along with their priorities p_T, where L is the trace length we use. A transition is of the form $\omega_s = (r_{s-1}, a_{s-1}, h_{s-1}, x_s, a_s, h_s, r_s, x_{s+1})$. Concretely, this consists of the following elements:

- r_{s-1}: reward at the previous time.
- a_{s-1}: action done by the agent at the previous time.
- h_{s-1}: recurrent state (in our case hidden state of the LSTM) at the previous time.
- x_s: observation provided by the environment at the current time.
- a_s: action done by the agent at the current time.
- h_s: recurrent state (in our case hidden state of the LSTM) at the current time.
- r_s: reward at the current time.
- x_{s+1}: observation provided by the environment at the next time.

The sequences never cross episode boundaries and they are stored into the buffer in an overlapping fashion, by an amount which we call the *replay period*. Finally, concerning the priorities, we followed the same prioritization scheme proposed by (Kapturowski et al., 2018) using a mixture of max and mean of the TD-errors in the sequence with priority exponent $\eta = 0.9$.

Evaluator: the evaluator shares the same network architecture as the learner, with weights that are fetched from the learner every episode. Unlike the actors, the experience produced by the evaluator is not sent to the replay buffer. The evaluator acts in the same way as the actors, except that all the computation is done within the single CPU process instead of delegating inference to the batch inference process. At the end of 5 episodes the results of those 5 episodes are average and reported. In this paper we report the average performance provided by such reports over the last 5% frames (for example, on Atari this is the average of all the performance reports obtained when the total frames consumed by actors is between 190M and 200M frames).

Learner: The learner contains two identical networks called the online and target networks with different weights θ and θ' respectively (Mnih et al., 2015). The target network's weights θ' are updated to θ every 400 optimization steps. θ is updated by executing the following sequence of instructions:

- First, the learner samples a batch of size 64 (batch size) of fixed-length sequences of transitions from the replay buffer, with each transition being of length L: $T_i = (\omega_s^i)_{s=t}^{t+L-1}$.
- Then, a forward pass is done on the online network and the target with inputs $(x_s^i, r_{s-1}^i, a_{s-1}^i, h_{s-1}^i)_{s=t}^{t+H}$ in order to obtain the state-action values $\{(Q(x_s^i, a; \theta), Q(x_s^i, a; \theta')\}$.
- With $\{(Q(x_s^i, a; \theta), Q(x_s^i, a; \theta')\}$, the $Q(\lambda)$ loss is computed.
- The online network is used again to compute the auxiliary contrastive loss.

- Both losses are summed (with by weighting the auxiliary loss by 0.1 as described in C), and optimized with an Adam optimizer.

- Finally, the priorities are computed for the sampled sequence of transitions and updated in the replay buffer.

A.2. V-MPO distributed setup

For on-policy training, we used a Podracer setup similar to (Hessel et al., 2021) for fast usage of experience from actors by learners.

TPU learning and acting: As in the Sebulba setup of (Hessel et al., 2021), acting and learning network computations were co-located on a set of TPU chips, split into a ratio of 3 cores used for learning for every 1 core used for inference. This ratio then scales with the total number of chips used.

Environment execution: Due to the size of the recurrent states used by CoBERL and stored on the host CPU, it was not possible to execute the environments locally. To proceed we used 64 remote environment servers which serve only to step multiple copies of the environment. 1024 concurrent episodes were processed to balance frames per second, latency between acting and learning, and memory usage of the agent states on the host CPUs.

A.3. Computation used

R2D2 We train the agent with a single TPU v2-based learner, performing approximately 5 network updates per second (each update on a mini-batch of 64 sequences of length 80 for Atari and 120 for Control). We use 512 actors, using 4 actors per CPU core, with each one performing ~ 64 environment steps per second on Atari. Finally for the batch inference process a TPU v2, which allows all actors to achieve the speed we have described.

V-MPO We train the agent with 4 hosts each with 8 TPU v2 cores. Each of the 8 cores per host was split into 6 for learning and 2 for inference. We separately used 64 remote CPU environment servers to step 1024 concurrent environment episodes using the actions returned from inference. The learner updates were made up of a mini-batch of 120 sequences, each of length 95 frames. This setup enabled 4.6 network updates per second, or 53.4k frames per second.

A.4. Complexity analysis

As stated, the agent consists of layers of convolutions, transformer layers, and linear layers. Therefore the complexity is $max\{O(n^2 \cdot d), O(k \cdot n \cdot d^2)\}$, where k is the kernel size in the case of convolutions, n is the size of trajectories, and d is the size of hidden layers.

B. Architecture description

B.1. Encoder

As shown in Fig. 1, observations O_i are encoded using an encoder. In this work, the encoder we have used is a ResNet-47 encoder. Those 47 layers are divided in 4 groups which have the following characteristics:

- An initial stride-2 convolution with filter size 3x3 ($1 \cdot 4$ layers).
- Number of residual bottleneck blocks (in order): $(2, 4, 6, 2)$. Each block has 3 convolutional layers with ReLU activations, with filter sizes 1x1, 3x3, and 1x1 respectively ($(2 + 4 + 6 + 2) \cdot 3$ layers).
- Number of channels for the last convolution in each block: $(64, 128, 256, 512)$.
- Number of channels for the non-last convolutions in each block: $(16, 32, 64, 128)$.
- Group norm is applied after each group, with a group size of 8.

After this observation encoding step, a final 2-layer MLP with ReLU activations of sizes $(512, 448)$ is applied. The previous reward and one-hot encoded action are concatenated and projected with a linear layer into a 64-dimensional vector. This 64-dimensional vector is concatenated with the 448-dimensional encoded input to have a final 512-dimensional output.

B.2. Transformer

As described in Section 2, the output of the encoder is fed to a Gated Transformer XL. For Atari and Control, the transformer has the following characteristics:

- Number of layers: 8.
- Memory size: 64.
- Hidden dimensions: 512.
- Number of heads: 8.
- Attention size: 64.
- Output size: 512.
- Activation function: GeLU.

For DmLab the transformer has the following characteristics:

- Number of layers: 12.

- Memory size: 256.
- Hidden dimensions: 128.
- Number of heads: 4.
- Attention size: 64.
- Output size: 512.
- Activation function: ReLU.

the GTrXL baseline is identical, but with a Memory size of 512.

B.3. LSTM and Value head

For both R2D2 and V-MPO the outputs of the transformer and encoder are passed through a GRU transform to obtain a 512-dimensional vector. After that, an LSTM with 512 hidden units is applied. The the value function is estimated differently depending on the RL algorithm used.

R2D2 Following the LSTM, a Linear layer of size 512 is used, followed by a ReLU activation. Finally, to compute the Q values from that 512 vector a dueling head is used, as in (Kapturowski et al., 2018), a dueling head is is used, which requires a linear projection to the number of actions of the task, and another projection to a unidimensional vector.

V-MPO Following the LSTM, a 2 layer MLP with size 512 and 30 (i.e. the number of levels in DMLab) is used. In the MLP we use ReLU activation. As we are interested in the multi-task setting where a single agent learns a large number of tasks with differing reward scales, we used PopArt (van Hasselt et al., 2016) for the value function estimation (see Table. 12 for details).

B.4. Critic Function

For DmLab-30 (V-MPO), we used a 2 layer MLP with hidden sizes 512 and 128. For Atari and Control Suite (R2D2) we used a single layer of size 512.

C. Hyperparameters

C.1. Atari and DMLab pre-processing

We use the commonly used input pre-processing on Atari and DMLab frames, shown on Tab. 8. One difference with the original work of (Mnih et al., 2015), is that we do not use frame stacking, as we rely on our memory systems to be able to integrate information from the past, as done in (Kapturowski et al., 2018). ALE is publicly available at `https://github.com/mgbellemare/Arcade-Learning-Environment`.

Hyperparameter	**Value**
Max episode length	30 min
Num. action repeats	4
Num. stacked frames	1
Zero discount on life loss	$false$
Random noops range	30
Sticky actions	$false$
Frames max pooled	3 and 4
Grayscaled/RGB	Grayscaled
Action set	Full

Table 7. Atari pre-processing hyperparameters.

C.2. Control Suite pre-processing

As mentioned in 4, we use no pre-processing on the frames received from the control environment.

C.3. DmLab pre-processing

Hyperparameter	**Value**
Num. action repeats	4
Num. stacked frames	1
Grayscaled/RGB	RGB
Image width	96
Image height	72
Action set	as in (Parisotto et al., 2020)

Table 8. DmLab pre-processing hyperparameters.

C.4. Control environment discretization

As mentioned, we discretize the space assigning two possibilities (1 and -1) to each dimension and taking the Cartesian product of all dimensions, which results in 2^n possible actions. For the `cartpole` tasks, we take a diagonal approach, utilizing each unit vector in the action space and then dividing each unit vector into 5 possibilities with the non-zero coordinate ranging from -1 to 1. The amount of actions this results in is outlined on Tab. 9.

C.5. Hyperparameters Search Range

The ranges we used to select the hyperparameters of CoBERL are displayed on Tab. 10.

C.6. Hyperparameters Used

We list all hyperparameters used here for completeness. Those in table 11 are used for all R2D2 experiments, including R2D2, R2D2-gTrXL, CoBERL, as well as CoBERL - loss. As out of the ones used in the search in C.5, they appeared to consistently be superior for these variants. Table 12 details the parameters used for the V-MPO DMLab-30 setting.

Task	**Action space size**	**Total amount of actions**
Acrobot	1	2
Cartpole	1	5
Cup	2	4
Cheetah	6	64
Finger	2	4
Fish	5	32
Pendulum	1	2
Reacher	2	4
Swimmer	5	32
Walker	6	64

Table 9. Control discretization action spaces.

Hyperparameter	**Value**
Q's λ	$\{0.8,\ 0.9\}$
Learning rate	$\{0.0001,\ 0.0003\}$
Contrastive loss weight	$\{0.01, 0.1, 1.0\}$

Table 10. Range of hyperparameters sweeps for R2D2.

Hyperparameter	**Value**
Optimizer	Adam
Learning rate	0.0003
Contrastive loss weight	0.1
Contrastive loss mask rate	0.15
Adam epsilon	10^{-7}
Adam beta1	0.9
Adam beta2	0.999
Adam clip norm	40
Q-value transform (non-DMLab)	$h(x) = sign(x)(\sqrt{\lvert x\rvert + 1} - 1) + \epsilon x$
Q-value transform (DMLab)	$h(x) = x$
Discount factor	0.997
Batch size	32
Trace length (Atari)	80
Trace length (non-Atari)	120
Replay period (Atari)	40
Replay period (non-Atari)	60
Q's λ	0.8
Replay capacity	80000 sequences
Replay priority exponent	0.9
Importance sampling exponent	0.6
Minimum sequences to start replay	5000
Target Q-network update period	400
Evaluation ϵ	0.01
Target ϵ	0.01

Table 11. Hyperparameters used in all experiments.

Hyperparameter	**Value**
Batch Size	120
Unroll Length	95
Discount	0.99
Target Update Period	50
Action Repeat	4
Initial η	1.0
Initial α	5.0
ϵ_η	0.1
ϵ_α	0.002
Popart Step Size	0.001
ReLIC Cost	1.0

Table 12. Hyperparameters used in V-MPO experiments.

D. Additional ablations

Table 13 shows the results of several gating mechanisms that we have investigated. As we can observe the GRU gate is a clear improvement especially on DMLab, only being harmful in median on the reduced ablation set of Atari games.

		CoBERL	'Sum' gate	'Concat' gate	w/o Gate
DMLab	Mean	$\mathbf{113.39\% \pm 3.64\%}$	$108.70\% \pm 3.23\%$	$106.31\% \pm 5.37\%$	$84.07\% \pm 5.71\%$
	Median	$\mathbf{112.02\%}$	108.95%	108.54%	104.33%
Atari	Mean	$\mathbf{698.0\% \pm 53.84\%}$	$548.66\% \pm 11.16\%$	$653.20\% \pm 59.13\%$	$591.33\% \pm 91.25\%$
	Median	276.6%	437.85%	325.96%	$\mathbf{320.09\%}$

Table 13. Gate ablations. Human normalized scores on Atari-57 ablation tasks and DMLab tasks. For the mean we include standard error over seeds.

E. Game scores

Atari (ablation games)	CoBERL	R2D2-gTrXL	R2D2	CoBERL-auxiliary loss
beam rider	22246.68 ± 7078.73	**61478.38 ± 27336.64**	34708.13 ± 11513.28	16318.78 ± 12438.73
enduro	2312.58 ± 35.59	2173.92 ± 135.85	**2346.15 ± 12.69**	2300.61 ± 75.25
breakout	**421.88 ± 1.50**	393.88 ± 31.14	336.19 ± 119.23	420.72 ± 9.86
pong	**21.00 ± 0.00**	**21.00 ± 0.00**	**21.00 ± 0.00**	**21.00 ± 0.00**
qbert	36932.00 ± 5498.71	21616.94 ± 3377.11	25129.37 ± 7139.03	**50362.28 ± 14109.22**
seaquest	167183.79 ± 112932.87	**326714.40 ± 51904.47**	124417.45 ± 128759.58	174867.68 ± 123876.30
space invaders	**34112.19 ± 10216.42**	21669.97 ± 6219.26	3712.64 ± 82.30	20192.85 ± 20815.81

Atari (ablation games)	CoBERL	CoBERL with LSTM before	CoBERL w/o LSTM
beam rider	22246.68 ± 7078.73	19233.70 ± 9849.79	**54345.65 ± 8111.23**
enduro	**2312.58 ± 35.59**	2304.59 ± 47.39	2227.61 ± 110.91
breakout	421.88 ± 1.50	**424.05 ± 6.83**	422.69 ± 7.58
pong	**21.00 ± 0.00**	21.00 ± 0.00	21.00 ± 0.00
qbert	**36932.00 ± 5498.71**	34773.82 ± 8972.64	33854.14 ± 5762.10
seaquest	**167183.79 ± 112932.87**	94254.91 ± 57966.74	151011.38 ± 93597.94
space invaders	**34112.19 ± 10216.42**	16980.01 ± 18410.59	4098.37 ± 938.44

Atari (ablation games)	CoBERL	No Skip Conn.	Sum gate	Concat
beam rider	22246.68 ± 7078.73	53379.23 ± 6229.05	**72882.42 ± 21239.63**	51371.38 ± 12560.94
enduro	**2312.58 ± 35.59**	2083.99 ± 382.91	2247.23 ± 82.22	2288.13 ± 58.59
breakout	**421.88 ± 1.50**	285.54 ± 181.62	403.36 ± 53.69	357.52 ± 72.63
pong	**21.00 ± 0.00**	**21.00 ± 0.00**	**21.00 ± 0.00**	**21.00 ± 0.00**
qbert	36932.00 ± 5498.71	31786.70 ± 5012.64	33807.00 ± 4294.93	**43487.35 ± 14518.46**
seaquest	167183.79 ± 112932.87	180119.87 ± 56159.52	294976.55 ± 41827.55	**399817.75 ± 36729.78**
space invaders	**34112.19 ± 10216.42**	27620.20 ± 17966.22	10387.59 ± 2858.63	20933.98 ± 13072.95

Atari (ablation games)	CoBERL with CURL	CoBERL with SimCLR
beam rider	25998.99 ± 18557.89	**27631.98 ± 31908.43**
enduro	2331.72 ± 36.46	**2344.57 ± 6.37**
breakout	328.38 ± 48.22	**381.42 ± 46.31**
pong	19.31 ± 2.38	**21.00 ± 0.00**
qbert	16073.04 ± 321.56	**18531.50 ± 3906.27**
seaquest	145340.79 ± 31778.00	**233181.04 ± 146617.39**
space invaders	21621.09 ± 15373.01	**28785.96 ± 18516.74**

Control Suite	CoBERL	gTrXL	R2D2
acrobot swingup	**359.75 ± 3.47**	215.39 ± 122.82	327.16 ± 5.35
fish swim	**624.40 ± 54.91**	91.32 ± 277.15	345.63 ± 227.44
fish upright	**942.33 ± 6.12**	849.52 ± 23.01	936.09 ± 11.58
pendulum swingup	**836.63 ± 9.77**	743.65 ± 52.44	831.86 ± 61.54
swimmer swimmer6	**447.60 ± 51.51**	225.97 ± 60.67	329.61 ± 26.77
finger spin	**985.05 ± 1.58**	977.41 ± 8.91	980.85 ± 0.67
reacher easy	**983.05 ± 2.47**	981.64 ± 1.99	982.28 ± 9.30
cheetah run	**525.06 ± 44.59**	115.15 ± 133.95	365.45 ± 50.40
walker walk	**780.54 ± 26.48**	595.96 ± 77.59	687.18 ± 18.15
ball in cup catch	978.28 ± 6.56	975.21 ± 1.77	**980.54 ± 1.94**
cartpole swingup	798.66 ± 7.72	**837.31 ± 4.15**	816.23 ± 2.93
cartpole swingup sparse	732.51 ± 18.60	747.94 ± 8.61	**762.57 ± 6.71**

Atari-57	CoBERL	R2D2-gTrXL	R2D2
alien	10229.89 ± 7932.26	9655.65 ± 1819.53	**10718.62 ± 4599.62**
amidar	2656.00 ± 966.37	**3883.37 ± 640.23**	2142.70 ± 241.96
assault	5469.38 ± 2607.77	10242.96 ± 4234.94	**13817.82 ± 1503.31**
asterix	**980283.74 ± 25765.32**	666449.34 ± 173208.21	724279.78 ± 195506.02
asteroids	**108985.96 ± 66922.29**	104932.39 ± 26450.55	74148.67 ± 49306.65
atlantis	**1091347.38 ± 37782.18**	979337.34 ± 8121.73	983110.27 ± 41978.78
bank heist	1117.91 ± 790.60	1318.51 ± 48.90	**1328.12 ± 456.18**
battle zone	77501.43 ± 39229.60	**98554.44 ± 43709.86**	94385.32 ± 13045.67
beam rider	22246.68 ± 7078.73	**61478.38 ± 27336.64**	34708.13 ± 11513.28
berzerk	**1756.21 ± 278.30**	626.60 ± 156.19	1466.54 ± 422.70
bowling	**184.32 ± 44.08**	42.33 ± 59.86	96.33 ± 30.66
boxing	**100.00 ± 0.00**	100.00 ± 0.00	99.71 ± 0.40
breakout	**421.88 ± 1.50**	393.88 ± 31.14	336.19 ± 119.23
centipede	66669.74 ± 6479.47	**76325.85 ± 16594.15**	74513.40 ± 12696.62
chopper command	**506146.56 ± 303260.36**	36993.73 ± 35157.10	33945.00 ± 13504.01
crazy climber	120806.63 ± 57107.29	120684.30 ± 6872.59	**157946.90 ± 42953.93**
defender	410044.34 ± 8847.63	293804.22 ± 72002.55	**462135.87 ± 12678.03**
demon attack	**137934.41 ± 5473.26**	131514.55 ± 9979.81	117580.02 ± 21157.45
double dunk	**24.00 ± 0.00**	19.93 ± 2.89	24.00 ± 0.00
enduro	2312.58 ± 35.59	2173.92 ± 135.85	**2346.15 ± 12.69**
fishing derby	**52.89 ± 2.89**	42.41 ± 11.36	49.92 ± 6.79
freeway	**34.00 ± 0.00**	34.00 ± 0.00	33.67 ± 0.47
frostbite	**8723.86 ± 1321.24**	7323.81 ± 2407.78	6909.40 ± 747.58
gopher	90684.67 ± 10244.87	**104482.57 ± 5853.11**	100203.22 ± 20908.66
gravitar	**6315.32 ± 223.45**	4282.56 ± 1705.21	5643.37 ± 631.75
hero	**20786.34 ± 139.43**	14010.16 ± 173.98	17996.97 ± 2841.71
ice hockey	19.82 ± 21.47	17.74 ± 11.55	**22.90 ± 10.41**
jamesbond	5576.58 ± 2595.65	**7962.54 ± 873.51**	7727.43 ± 2489.21
kangaroo	12173.70 ± 3129.91	13520.79 ± 2068.49	**14436.53 ± 116.90**
krull	37813.60 ± 18826.63	**58459.18 ± 38117.13**	12285.18 ± 572.14
kung fu master	**126648.43 ± 5685.80**	70772.67 ± 28598.85	102387.20 ± 17781.04
montezuma revenge	**833.33 ± 1178.51**	0.00 ± 0.00	133.33 ± 188.56
ms pacman	**11295.44 ± 4623.95**	11146.67 ± 902.33	9893.29 ± 1172.58
name this game	25044.23 ± 6659.51	**26944.23 ± 1315.96**	24348.53 ± 1917.48
phoenix	**514890.69 ± 169407.85**	322625.85 ± 92449.70	194688.45 ± 178633.63
pitfall	**0.00 ± 0.00**	0.00 ± 0.00	0.00 ± 0.00
pong	**21.00 ± 0.00**	**21.00 ± 0.00**	**21.00 ± 0.00**
private eye	10520.09 ± 4986.07	**15057.95 ± 42.35**	4988.65 ± 6938.00
qbert	**36932.00 ± 5498.71**	21616.94 ± 3377.11	25129.37 ± 7139.03
riverraid	24894.12 ± 3079.76	23368.64 ± 2389.28	**28057.87 ± 576.75**
road runner	279422.07 ± 223362.40	**518566.46 ± 62912.72**	0.00 ± 0.00
robotank	**82.36 ± 8.68**	50.70 ± 33.29	67.41 ± 21.91
seaquest	167183.79 ± 112932.87	**326714.40 ± 51904.47**	124417.45 ± 128759.58
skiing	-29958.52 ± 2.39	**-22984.50 ± 9882.73**	-29963.55 ± 1.82
solaris	**4931.47 ± 2344.05**	1661.77 ± 1032.29	2610.71 ± 1573.21
space invaders	**34112.19 ± 10216.42**	21669.97 ± 6219.26	3712.64 ± 82.30
star gunner	**106292.95 ± 29670.23**	104395.14 ± 16821.38	93412.15 ± 10284.46
surround	**9.30 ± 0.50**	9.25 ± 0.54	8.78 ± 0.50
tennis	**15.61 ± 11.17**	8.00 ± 11.31	0.00 ± 0.00
time pilot	**39261.92 ± 7400.70**	14303.91 ± 1695.58	23611.05 ± 3357.51
tutankham	38.79 ± 49.73	23.11 ± 16.91	**84.30 ± 46.88**
up n down	397836.59 ± 111520.35	289177.29 ± 135467.48	**422332.40 ± 41201.70**
venture	**1873.20 ± 30.56**	1782.00 ± 83.88	1640.10 ± 141.12
video pinball	**276228.36 ± 133392.82**	58865.88 ± 51845.02	206756.24 ± 72958.76
wizard of wor	**18707.03 ± 21128.79**	14862.25 ± 8267.18	16548.31 ± 10862.27
yars revenge	**322255.03 ± 171716.17**	201576.03 ± 23183.30	316415.89 ± 176059.44
zaxxon	14420.27 ± 10309.48	20132.38 ± 3941.10	**31116.74 ± 1966.53**

DmLab Levels	coberl	gtrxl
rooms collect good objects test	$\mathbf{9.75 \pm 0.12}$	9.67 ± 0.22
rooms exploit deferred effects test	53.56 ± 4.01	$\mathbf{54.87 \pm 0.87}$
rooms select nonmatching object	$\mathbf{64.04 \pm 10.51}$	58.54 ± 4.07
rooms watermaze	$\mathbf{58.35 \pm 1.65}$	43.36 ± 7.11
rooms keys doors puzzle	27.12 ± 5.62	$\mathbf{34.42 \pm 8.53}$
language select described object	617.55 ± 0.72	$\mathbf{624.50 \pm 14.14}$
language select located object	$\mathbf{614.88 \pm 3.14}$	578.00 ± 2.82
language execute random task	$\mathbf{195.33 \pm 66.18}$	191.70 ± 15.61
language answer quantitative question	$\mathbf{330.66 \pm 5.03}$	293.00 ± 9.89
lasertag one opponent large	$\mathbf{14.16 \pm 4.48}$	10.50 ± 4.24
lasertag three opponents large	26.83 ± 4.48	$\mathbf{28.83 \pm 5.89}$
lasertag one opponent small	$\mathbf{29.83 \pm 1.52}$	27.75 ± 3.88
lasertag three opponents small	$\mathbf{46.0 \pm 1.40}$	35.0 ± 0.01
natlab fixed large map	$\mathbf{44.66 \pm 7.28}$	38.66 ± 13.85
natlab varying map regrowth	$\mathbf{28.33 \pm 6.93}$	21.87 ± 11.13
natlab varying map randomized	$\mathbf{45.88 \pm 9.02}$	32.08 ± 12.31
platforms hard	$\mathbf{57.53 \pm 7.17}$	35.00 ± 25.92
platforms random	73.45 ± 2.64	$\mathbf{86.03 \pm 0.45}$
psychlab arbitrary visuomotor mapping	$\mathbf{57.38 \pm 4.14}$	32.96 ± 10.55
psychlab continuous recognition	51.36 ± 1.94	$\mathbf{58.02 \pm 0.71}$
psychlab sequential comparison	28.11 ± 1.09	$\mathbf{31.83 \pm 0.94}$
psychlab visual search	$\mathbf{79.91 \pm 0.14}$	79.58 ± 0.58
explore object locations small	$\mathbf{83.71 \pm 6.87}$	74.10 ± 7.10
explore object locations large	$\mathbf{61.16 \pm 2.11}$	61.12 ± 2.65
explore obstructed goals small	$\mathbf{260.37 \pm 6.97}$	234.16 ± 10.60
explore obstructed goals large	$\mathbf{114.44 \pm 2.45}$	76.25 ± 8.83
explore goal locations small	$\mathbf{370.00 \pm 2.50}$	339.10 ± 3.43
explore goal locations large	$\mathbf{142.50 \pm 9.34}$	127.50 ± 14.14
explore object rewards few	$\mathbf{76.93 \pm 13.61}$	70.23 ± 0.61
explore object rewards many	$\mathbf{105.80 \pm 3.37}$	95.50 ± 3.53

F. Licenses

The The Arcade Learning Environment (Bellemare et al., 2013) is released as free, open-source software under the terms of the GNU General Public License. The latest version of the source code is publicly available at: http://arcadelearningenvironment.org

DeepMind Control Suite (Tassa et al., 2018) iis released as free, open-source software under the terms of Apache-2.0 License. The latest version of the source code is publicly available at: `https://github.com/deepmind/dm_control/blob/master/dm_control/suite/README.md`

DmLab (Beattie et al., 2016) is released as free, open-source software under the terms of Apache-2.0 License. The latest version of the source code is publicly available at: `https://github.com/deepmind/lab/tree/master/game_scripts/levels/contributed/dmlab30`