Efficient long term prediction of pixels with the Modular Autoencoder for Atari games

Adrian Millea
a.millea14@imperial.ac.uk
Department of Computing, Imperial College London, SW7 2AZ, London

Abstract

When working with pixel data as input for reinforcement learning, the compactness of the inner representation of the agent is critical for subsequent efficient learning, be it policy learning or state space structuring (e.g. subgoal discovery). We strive to provide such a representation, where we semantically decompose the images into static objects, moving objects and controlled moving objects, each modelled separately and then combined to reconstruct the image and perform long term predictions in pixel space. We combine multiple existing ideas to form a new type of autoencoder, closely based on the original VAE [Kingma and Welling, 2013]. We call this the Modular Autoencoder and show that it has a number of advantages like efficiency of learning and it enables a much lower dimensionality of the highly compressed representation.

Keywords: **deep reinforcement learning, temporal autoencoder, static objects, contingency awareness**

1 Introduction

In Deep Reinforcement Learning[1] (DRL) we almost always work with pixel data, which is very high dimensional, but also very redundant, meaning it does not convey a lot of information. A significant number of pixels constitute static textures, which can be easily represented by a very low dimensional space without any loss of information, in other words losslessly compressed. In RL we would like to map states to actions, in an optimal manner, such that we maximize return (we don't give here the description of the RL problem, we assume the reader is well informed on this, see for a brief description the first DRL paper [Mnih et al., 2015]) We strive first to isolate these static objects such that we can find a good, low dimensional representation for them and then later, after they can be reconstructed to a high degree of accuracy, consider also the moving objects for the reconstruction objective. Moreover, we want to differentiate between uncontrolled moving objects and agent-controlled moving objects, or in other words enable the agent to be contingency aware [Bellemare et al., 2012]. In general, the first type of moving objects in Atari games (we will exclusively work with Atari games in this paper) obeys simple rules, these objects can be modelled by simple low-dimensional dynamical models. The second type will take as input also the action performed by the agent in the previous timestep. In this way we succinctly and semantically represent an image, close to how humans do it. The purpose of this decomposition is to be able to have an overall low-dimensional representation of the environment in which a DRL agent acts as well as meaningful one such that dynamical models can be efficiently learned for moving objects to a high degree of accuracy that will enable long term prediction in pixel space. The necessity for low-dimensionality is obvious for policy learning as well as model learning. We hope that by reducing the dimensionality of the representation needed, we will significantly improve on the learning speed of a DRL agent.

1.1 Problem statement

Even with imperfect models [Weber et al., 2017] which feed into a policy network, model-based techniques have very good performance, especially in terms of data efficiency [Deisenroth and Rasmussen, 2011]. The models can be used for exploration, can be used to predict the values of future states, without the need for the actual interaction with the environment and generally provide the agent with more

[1] `https://deepmind.com/blog/deep-reinforcement-learning/`

knowledge and flexibility (by learning or changing models as a function of the context, or new data) in its behaviour. The main burden of model based RL (MB-RL) is the high dimensionality of the models which incur a high computational burden. However, we point out that the actual moving entities in an Atari game (in general that's why we need to model the environment, for the moving entities, conditioned on the action or independent of it) are quite simple in terms of their dynamics and could even be modelled easily as a 1 dimensional time-series. Having these simple and accurate models of the moving entities in an image, we could then combine them to make predictions of the whole state and thus predict the value of some future state to guide the agent's behaviour. Assuming we want to still use the deep network distributed representation paradigm in which we do not segment entities by some preprocessing algorithm but let them be learned by the network, still some problems remain:

1. How can we figure out the representation of the moving entities in the environment in the inner dynamics of the network?
2. How can we rapidly learn simple models to predict future dynamics of objects? and
3. How to flexibly combine these models for an accurate prediction of whole frames?

We show that a simple solution to this is to use an autoencoder which has a few minor modifications which constrain the inner representation as we need it, i.e. it separates the static and moving objects in the most inner layer of the autoencoder.

2 Modular Autoencoder

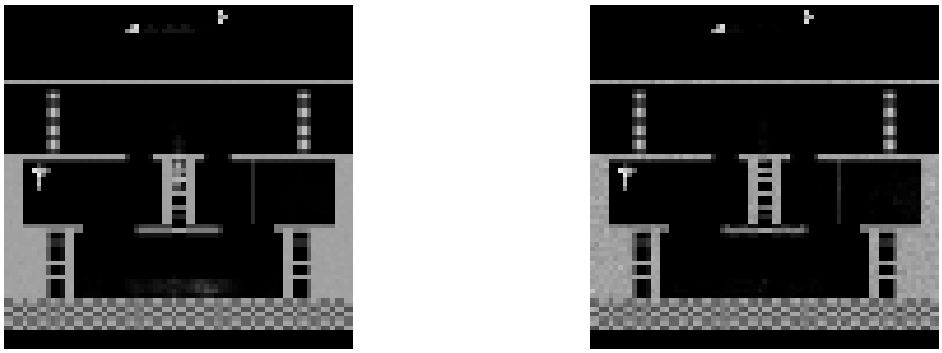
A meaningful and compressed representation will enhance similarity computation (assuming we would like to compare states, which is usually done for state space clustering) as well as performance of policy and value learning as they are given already high-level semantic features from the environment. To this end we are looking for a transformation to give us a minimal state representation. As we're interested in a meaningful differentiation between the states, looking at what is changing between states should be more informative than the bulk, sort of useless data, given by static textures. We thus propose to look at the loss function for reconstruction over time. When trying to reconstruct a single frame, the loss of the autoencoder is given by:

$$\begin{aligned} L = ||\mathbf{x} - (\psi \circ \phi)\mathbf{x}||^2 \text{ with} \\ \phi : \mathcal{X} \to \mathcal{F} \text{ the encoder, and} \\ \psi : \mathcal{F} \to \mathcal{X} \text{ the decoder} \end{aligned}$$

where $\mathcal{X}$ is the input space and $\mathcal{F}$ is the feature space, the inner representation of the autoencoder. If we want our autoencoder to compress static objects and textures as much as possible, this suggests a procedure where we first look at and try to reconstruct just the static objects and textures in the image, and after this is compressed enough, we consider the moving objects as well. This leads us to look at the input between adjacent time steps. Consider the input at time t, to be $\mathbf{x}^t$, we first augment the autoencoder with a simple change detector, $\mathbf{d}^t = (\mathbf{x}^{t+1} - \mathbf{x}^t)^2$. We will subtract this $\mathbf{d}^t$ from the desired reconstructed frame (the target), basically ignoring this change when backpropagating the gradients. This gives the new loss at time $t + 1$:

$$L_{t+1} = ||\mathbf{x}^{t+1} - \mathbf{d}^t - (\psi \circ \phi)\mathbf{x}^{t+1}||^2$$

This is the first part of training where we try to reconstruct the static entities and textures. We use a subset of the inner neurons for this procedure, $\mathbf{z}_s$ and the weights associated with them. After we have a good fit we fix these weights and allow the other set of weights $\mathbf{z}_m$, that were fixed initially, to adapt to model the added second loss $\mathbf{d}^t$. In this way we actually minimize the original autoencoder reconstruction loss but we partition the entire set of weights into 2 sets (for now), one that adapts



(a) Default loss. (b) Modified loss.

Figure 1: The two reconstructions given by the two different losses on Montezuma's Revenge.

to static objects and the other on moving objects. We note that we do not differentiate yet between controlled and uncontrolled moving objects, but keep in mind that $\mathbf{d^t} = \mathbf{d}_u^t + \mathbf{d}_c^t$, where $\mathbf{d}_u^t$ denotes the uncontrolled entities and $\mathbf{d}_c^t$ denoted the controlled ones.

We test this loss function to see if the autoencoder reconstructs indeed the static images. And we see in Figure 1 that it does. We see two reconstructions for the default loss and the modified loss, where for the default loss one can still see the path of the ghost, as well as traces of the agent on the middle stairs. Some vague traces for the number of lives can also be observed, while for the modified loss we cannot see any traces for moving objects. The actions chosen in the frames shown are uniformly random, and the autoencoder used is a vanilla VAE. The autoencoder is run just for a few epochs which is sufficient to see the difference between the two losses. Other versions of spatio-temporal autoencoders exist, for example see [Patraucean et al., 2015], however they suffer from a number of shortcomings. For example, even though it uses Convolutional LSTMs, it uses an additional prediction module, which generates a map with one 2D vector per pixel, denoting the displacement in the two directions. This is already quite expensive, and in addition, the gradient of this transformation needs to be computed and backpropagated which increases computational cost even more. Moreover, [Patraucean et al., 2015] and [Oh et al., 2015] both use the pixel space as a unified space and prediction does not differentiate between static and moving objects. This is exactly what we set out to do. We will combine the ideas from VAE [Kingma and Welling, 2013], [Oh et al., 2015] and [Patraucean et al., 2015] into a single combined representation and training methodology. Formally, let $\mathbf{z}_s$ the set of latent variables that represents *static objects*, $\mathbf{z}_m$ the set of latents corresponding to the *moving objects* and $\mathbf{z}_c$ the set of latents corresponding to the *controlled moving objects*, so the whole set of latent variables is $\mathbf{z} = \{\mathbf{z}_s, \mathbf{z}_m, \mathbf{z}_c\}$. Thus, the conditional probability of an observation image $\mathbf{x}$ is given by $p_\theta(\mathbf{x}|\mathbf{z}_s, \mathbf{z}_m, \mathbf{z}_c)$, where θ are the parameters of the autoencoder (originally these were the same as the variational parameters of the decoder, however we will see that in our work they will be different). The posterior probability of the latents is given as:

$$p(\mathbf{z}_s, \mathbf{z}_m, \mathbf{z}_c|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z}_s, \mathbf{z}_m, \mathbf{z}_c)p(\mathbf{z}_s, \mathbf{z}_m, \mathbf{z}_c)}{p(\mathbf{x})}$$

The integral $p(\mathbf{x}) = \int p(\mathbf{z})p(\mathbf{x}|\mathbf{z})d\mathbf{z}$ is intractable as well as the posterior above. This is why the AEVB algorithm comes in handy, to learn parameters θ of $p(\mathbf{x}|\mathbf{z})$ and variational parameters ϕ of the approximating distribution $q_\phi(\mathbf{z}|\mathbf{x}) \approx p(\mathbf{z}|\mathbf{x})$.

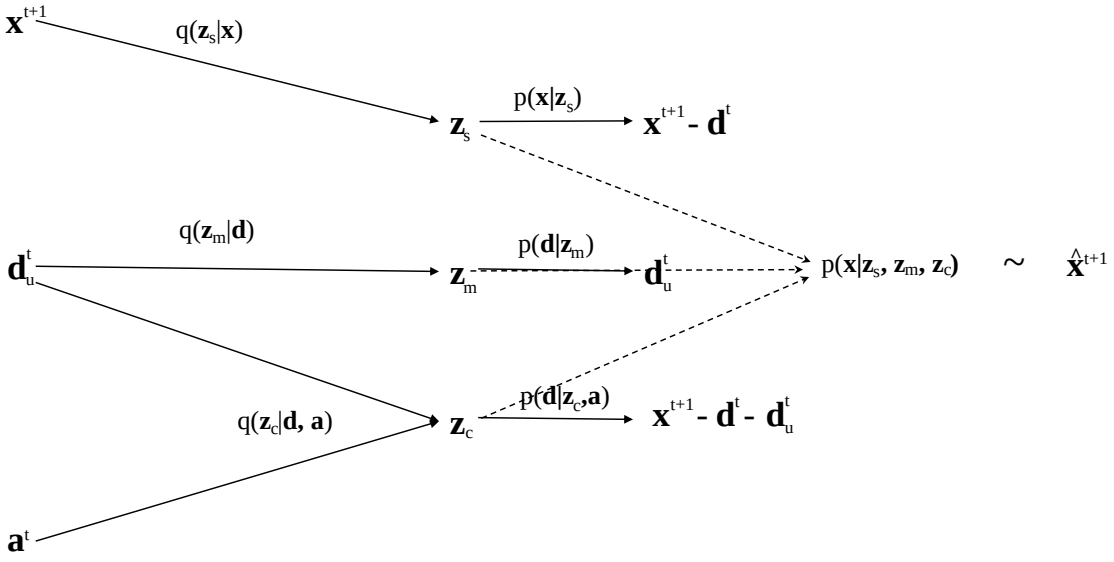


Figure 2: A depiction of the modular autoencoder.

Decoder Similar to [Kingma and Welling, 2013] we use for the decoder the following model:

$$\begin{aligned} \log p(\mathbf{x}|\mathbf{z}_s, \mathbf{z}_m, \mathbf{z}_c) &= \log \mathcal{N}(\mathbf{x}|\mu, \sigma^2\mathbf{I}) \\ \text{with } \mu &= \mathbf{W}_2\mathbf{h} + \mathbf{b}_2 \\ \log \sigma^2 &= \mathbf{W}_3\mathbf{h} + \mathbf{b}_3 \\ \text{and } \mathbf{h} &= tanh(\mathbf{W}_1[\mathbf{z}_s; \mathbf{z}_m; \mathbf{z}_c] + \mathbf{b}_1) \end{aligned}$$

We cannot factorize the decoder function, as the input image needs to be reconstructed from all the latent variables, we cannot reconstruct the image separately from each individual set of latents, this would not make sense. Thus we use as input to the decoder the concatenation of the different sets of latent variables $[\mathbf{z}_s; \mathbf{z}_m; \mathbf{z}_c]$.

2.1 Training regimes and Encoders

A critical part in how everything fits together are the different training regimes. We need to differentiate somehow between the static, moving and controlled moving objects, such that the different set of latent variables model the semantically different entities in the image. Thus the three encoder models will look a bit different. First we note that we have three different inputs to our encoders, $\mathbf{x}$, $\mathbf{d}$ and $\mathbf{a}$ (we omit the time indices in this initial description, but we note that they are there). The full encoder model is given by:

$$p(\mathbf{z}_s, \mathbf{z}_m, \mathbf{z}_c|\mathbf{x}, \mathbf{d}, \mathbf{a}) = p(\mathbf{z}_s|\mathbf{x})p(\mathbf{z}_m|\mathbf{d})p(\mathbf{z}_c|\mathbf{d}, \mathbf{a})$$

where we see that $\mathbf{z}_s$ depends only on the input $\mathbf{x}$, and not on the difference between subsequent inputs $\mathbf{d}$, or the action $\mathbf{a}$, $\mathbf{z}_m$ depends only on $\mathbf{d}$ and $\mathbf{z}_c$ depends on $\mathbf{d}$ and $\mathbf{a}$. We depict these dependencies in Figure 2.

2.1.1 Encoding the static textures

In Atari games, a single level has very little variation of the static textures, which means we could easily model the transformation for one level with just one latent neuron in the inner layer of the autoencoder, but since we would like to be able to model all levels of one game, assuming the inner neurons would be binary we would be able to model n levels with $\log_2 n$ neurons, but since we have real-valued neurons this number should be smaller. We thus choose $\mathbf{z}_s \in \mathbb{R}^{\log_{10} n}$, with n the number of possible levels in the game, to balance between the efficiency of representation and the efficiency of learning. Assuming we want to have the same autoencoder to model multiple games, then $\mathbf{z}_s \in \mathbb{R}^{\log_{10} n \times m}$ with m the number of games. As we mentioned and showed before a good reconstruction error for the static entities is:

$$L_s^{t+1} = ||\mathbf{x}^{t+1} - \mathbf{d}^t - \widehat{\mathbf{x}}^{t+1}||^2$$

where $\widehat{\mathbf{x}} \sim p(\mathbf{x}|\mathbf{z}_s, \mathbf{z}_m, \mathbf{z}_c)$ is sampled from the decoder model, as shown in 2. In this step we take random actions such that the representation does not catch the moving controlled entities as static ones. From a human perspective, this an understanding phase, where one observes what is moving and what is not such that in the next step one knows what should be considered as moving objects. After the model reaches a reasonable reconstruction error, these weights remain fixed for the next training phases. When a new level is seen the weights start adapting again to model this as well, however the training is interleaved with sampling from the previously seen levels, such that the knowledge learned is not forgotten (this is an ongoing known problem in the literature also known as catastrophic forgetting [Kirkpatrick et al., 2017, Lee et al., 2017] and can be overcome in a number of ways, we adopt one of the simplest solutions, by retraining on previous experiences, since the difference between the levels is quite small). For the static objects, the latent variables have the standard form, as in the original VAE[Kingma and Welling, 2013], so we skip their description.

2.1.2 Encoding the moving entities

For this step, we fix the controlled moving objects, meaning we choose the action that is equivalent to doing nothing. This is not an unreasonable assumption to make, it means the agent first observes what is happening, what is moving, and learns how to predict and encode the different entities in the image before deciding to take some motor action and figuring out which are the entities under its control. This means that in the decomposition $\mathbf{d}^t = \mathbf{d}_u^t + \mathbf{d}_c^t$, we have $\mathbf{d}_c^t = 0$, since there is no change associated with controlled objects, they are static in this regime. If we would also use convolutions on these input, the convolutional model should not be translation invariant (meaning we apply each filter on each patch and then do max pooling), but translation equivariant (avoid max pooling, so we are able to differentiate between the positions of the entities). Moreover, endowing the output of the convolutions with some spatial indices would easily enable the detection of moving objects (e.g. if adjacent positions detect the same object in two subsequent timesteps, it means the object is moving) and prediction would simply amount to learning the patterns of these indices, e.g. simple curve fitting. So the reconstruction error for the moving entities is:

$$L_m^{t+1} = ||\mathbf{d}_u^t - \widehat{\mathbf{d}}_u^t||^2$$

with $\widehat{\mathbf{d}}^t \sim q(\mathbf{d}|\mathbf{z}_m)$ is sampled from the respective decoder. After reasonable reconstruction error, these learned weights remain fixed for the rest of the training. For moving objects we would like to use some dynamic model, for example a neural network with memory, an LSTM [Hochreiter and Schmidhuber, 1997], or we could use a modified LSTM, with convolutions instead of normal multiplication as in [Patraucean et al., 2015] or a simple RNN (we choose this option since the models should be quite simple, the entities do not have complex moving patterns, similar to [Fabius and van Amersfoort, 2014]). The model at time t

is then given by:

$$\begin{aligned}\log q(\mathbf{z}_m|\mathbf{d}_u^t) &= \log \mathcal{N}(\mathbf{z}_m|\mu_m, \sigma_c^2\mathbf{I}) \\ \text{with } \mu_m &= \mathbf{W}_4\mathbf{h}_m^t + \mathbf{b}_4 \\ \log \sigma_m^2 &= \mathbf{W}_5\mathbf{h}_m^t + \mathbf{b}_5 \\ \text{and } \mathbf{h}_m^t &= tanh(\mathbf{W}_6\mathbf{d}_u^t + \mathbf{W}_7\mathbf{h}_m^{t-1} + \mathbf{b}_6)\end{aligned}$$

2.1.3 Encoding the controlled moving objects

Having already learned a representation for the static and uncontrolled moving objects we now strive to learn what the agent controls (in the case of Atari games, this amounts to the agent body and any possible bullets he might fire). In the first training regime we ignored the agent as it was not static (we were taking random moves) and in the second training regime the agent was static (we took the action equivalent to doing nothing at every step) so we avoided considering the controlled moving objects in both cases. Thus, in this third regime we would like to reconstruct what is left from the whole input image, i.e.:

$$\mathbf{x}^{t+1} - \widehat{\mathbf{x}}^{t+1} - \widehat{\mathbf{d}}_u^t$$

assuming the previous steps have managed to minimize their respective losses, this means $\widehat{\mathbf{x}}^{t+1} \approx \mathbf{x}^{t+1} - \mathbf{d}^t$, with $\mathbf{d}^t = \mathbf{d}_u^t + \mathbf{d}_c^t$ and $\widehat{\mathbf{d}}_u^t \approx \mathbf{d}_u^t$. Replacing terms we get:

$$\mathbf{x}^{t+1} - (\mathbf{x}^{t+1} - \mathbf{d}_u^t - \mathbf{d}_c^t) - \mathbf{d}_u^t = \mathbf{d}_c^t$$

exactly what we wanted to reconstruct, the difference given by the controlled moving objects. So we see that by summing up all these losses we actually get the original autoencoder loss, however with a twist, now we have semantical modularity, we have partitioned the set of latent variables into 3 distinct meaningful sets, which model static, uncontrolled and controlled moving entities.

For the action dependence, the encoder model takes into account the action through multiplicative interaction, just as in [Oh et al., 2015]. We can use a 3-way tensor to model the interaction between the input and the action, however as stated in the original work, this is expensive in practice, thus we choose to use 2 matrices, assuming there are already enough parameters to fit (in the original work they use 3) and we just need to match dimensions such that the multiplications are valid. The model for the controlled moving objects is as follows:

$$\begin{aligned}\log q(\mathbf{z}_c|\mathbf{d}_c^t, \mathbf{a}^t) &= \log \mathcal{N}(\mathbf{z}_c|\mu_c, \sigma_c^2\mathbf{I}) \\ \text{with } \mu_c &= \mathbf{W}_8\mathbf{h}_c^t + \mathbf{b}_8 \\ \log \sigma_c^2 &= \mathbf{W}_9\mathbf{h}_c^t + \mathbf{b}_9 \\ \text{and } \mathbf{h}_c^t &= tanh(\mathbf{W}_{10}\mathbf{d}_c^t \odot \mathbf{W}_{11}\mathbf{a}^t + \mathbf{W}_{12}\mathbf{h}_c^{t-1} + \mathbf{b}_{10})\end{aligned}$$

where $\odot$ is element wise multiplication. We see that if $\mathbf{d}^t \in \mathbb{R}^n$, and $|\mathcal{A}| = m$, then $\mathbf{W}_9 \in \mathbb{R}^{f\times n}$ and $\mathbf{W}_{10} \in \mathbb{R}^{f\times m}$. We see that the models are quite simple, similar to the original VAE and they enable semantically meaningful sets of latent variables, modularity of learning and increased efficiency of representation and learning. The loss over T timesteps is similar to the one in [Oh et al., 2015], however we note that this is a general form for optimizing over time. However, for the overall loss over time, we would like to incrementally optimize for each timestep, and only when one has achieved sufficient performance, advance to the next one, as:

$$L_T^K = \frac{1}{2T}\sum_{k=1}^{T}\sum_{t=1}^{T}||\mathbf{x}_{t+k} - (\psi \circ \phi)\mathbf{x}_{t+k}||^2$$

We see that when looking at the joing prior $p(\mathbf{z}_s, \mathbf{z}_m, \mathbf{z}_c)$, this can be easily factorized as $p(\mathbf{z}_s)p(\mathbf{z}_m)p(\mathbf{z}_c)$ as the latent variables are independent of each other. We know that $\mathbf{z}_c$ is dependent on the action, however we are not considering the policy here, i.e. $p(\mathbf{a}|\mathbf{x})$, so even though $\mathbf{a}$ is a random variable governed by some policy, we consider it as an input, so fixed, through the multiplicative interaction. As in the original VAE, the priors over the latents are isotropic Gaussians $p(\mathbf{z}_i) \sim \mathcal{N}(\mathbf{z}_i|0, \mathbf{I})$, with $\mathbf{z}_i = \{\mathbf{z}_s, \mathbf{z}_m, \mathbf{z}_c\}$.

So we can now try to answer the questions asked at the beginning 1.1:

1. By forcing different losses on different sets of latent variables. For the controlled objects add the action as an input to the model.

2. By having smaller sets of latent variables, and smaller inputs to the models that deal with moving entities. We note that $\mathbf{d}^t$ will be highly sparse, compared to $\mathbf{x}^t$.

3. By concatenating all the sets of latents and feed them as input to the decoder $p(\mathbf{x}|\mathbf{z})$.

References

[Bellemare et al., 2012] Bellemare, M. G., Veness, J., and Bowling, M. (2012). Investigating contingency awareness using atari 2600 games.

[Deisenroth and Rasmussen, 2011] Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472.

[Fabius and van Amersfoort, 2014] Fabius, O. and van Amersfoort, J. R. (2014). Variational recurrent auto-encoders. *arXiv preprint arXiv:1412.6581*.

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

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[Kirkpatrick et al., 2017] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, page 201611835.

[Lee et al., 2017] Lee, S.-W., Kim, J.-H., Jun, J., Ha, J.-W., and Zhang, B.-T. (2017). Overcoming catastrophic forgetting by incremental moment matching. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 4655–4665. Curran Associates, Inc.

[Mnih et al., 2015] 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. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[Oh et al., 2015] Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. (2015). Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, pages 2863–2871.

[Paszke et al., 2017] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

[Patraucean et al., 2015] Patraucean, V., Handa, A., and Cipolla, R. (2015). Spatio-temporal video autoencoder with differentiable memory. *arXiv preprint arXiv:1511.06309*.

[Weber et al., 2017] Weber, T., Racanière, S., Reichert, D. P., Buesing, L., Guez, A., Rezende, D. J., Badia, A. P., Vinyals, O., Heess, N., Li, Y., et al. (2017). Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203*.