General Instructions (summarized, to be removed before submission)

- Structure — article-like, presenting the findings of our work clearly and objectively, provide well-supported analysis backing up our claims with references or empirical/theoretical evidence where appropriate
- Length — 4 to 8 pages, excluding references and appendices.
- Focus — provide insights valuable for future researchers or practitioners looking to reproduce / build upon the paper
- Evaluation — not solely based on quantitative outcomes, but rather on depth, quality of analysis and clarity of the report

ATML Report

Jonatan Bella *jonatan.bella@usi.ch*
Alessia Berarducci *alessia.berarducci@usi.ch*
Jonas Knupp *jonas.knupp@usi.ch*
Tobias Erbacher *tobias.erbacher@usi.ch*

Abstract

In this project we aim to analyze the claims of Mahankali et al. (2024a) in the paper Random Latent Exploration for Deep Reinforcement Learning[1] which introduces a new technique to explore the state space and thus yield better overall agent scores. Moreover, we intend to reproduce the results obtained by the authors, expand on the findings and verify that the results are not cherry-picked. <write the main findings of our project> This report is prepared as part of the course project in *Advanced Topics in Machine Learning* at *Università della Svizzera italiana* in the autumn semester of 2024.

1 Introduction

The paper we investigate deals with the problem of motivating an agent in a high-dimensional state space to explore the environment more exhaustively during training and thereby find non-obvious trajectories that can lead to higher long-term rewards for both discrete and continuous action spaces. The paper compares the new *Random Latent Exploration* (RLE) technique to standard *Proximal Policy Optimization* (PPO, see Schulman et al. (2017)), *NoisyNet* (see Fortunato et al. (2018)) and *Random Network Distillation* (RND, see Burda et al. (2019)).

Exploration is one of the major issues of Reinforcement Learning (RL) and its techniques can generally be divided into two categories: Noise-based exploration and bonus-based exploration. There are advantages and disadvantages for both of them but we always have to keep in mind that always choosing the highest short-term (local) reward does not necessarily also yield the highest long-term (global) reward, e.g. picture the environment as shown in figure 1a. Here, the agent starting in the blue state will always choose the

[1] Please note that there exists an older version of this paper by Mahankali et al. (2024b) marked in the references with "(OLD VERSION)" which was provided by the course instructors. Only where there are significant differences we will refer to the new version.

small reward of 1, i.e. going right, then move back to the initial state, then move right, and so on, dithering between these two states, instead of accepting to collect a reward of 0 by going left in order to be able to collect the much higher reward of 100 in the following step.

Mahankali et al. (2024a) use RND to represent bonus-based exploration, NoisyNet to represent noise-based exploration and standard PPO as the baseline benchmark.

(a) Pure rewards.

(b) Rewards with randomization (example).

Figure 1: An exemplary environment consisting of four state where transitions can occur between neighboring states. The blue node is the initial state and the numbers are the rewards.

1.1 Random Latent Exploration

The new idea that the authors Mahankali et al. (2024a) present is to augment the reward function by adding a randomized term which incentivizes the agent to explore a larger portion of the state space. In particular, we will call this term $F(s, z)$ or intrinsic reward function, where $s \in \mathcal{S}$ is any state of the state space $\mathcal{S}$, and $\mathbf{z} \in \mathbb{R}^d$ is a d-dimensional vector sampled from a given distribution $P_{\mathbf{z}}$. The authors Mahankali et al. (2024a) claim that RLE's efficiency is not significantly affected by the choice of $P_{\mathbf{z}}$. If at time step $t \in \{0, 1, 2, ..., T\}$ the agent in state s_t takes action $a_t \sim \pi(.|s_t)$ from policy π, then it will obtain the task reward $r(s_t, a_t)$. In RLE, we train a so-called latent-conditioned policy network $\pi(.|s, \mathbf{z})$ and latent-conditioned value network $V^\pi(s, \mathbf{z})$ to estimate and then maximize the expected sum of rewards from a given state, aware of the random term z:

$$V^\pi(s, \mathbf{z}) \approx \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \left(r(s_t, a_t) + F(s_{t+1}, \mathbf{z}) \right) \right] \tag{1}$$

In equation 1, γ describes the discount factor and $F(s_{t+1}, \mathbf{z}) = \boldsymbol{\phi}(s) \cdot \mathbf{z}$ where $\boldsymbol{\phi}(s) : \mathcal{S} \to \mathbb{R}^d$ is a feature extraction network. The feature network is updated as a linear combination of the old feature network's weights and the weights of the value network[2]. Both $\boldsymbol{\phi}$ and $\mathbf{z}$ are d-dimensional vectors. Note that although equation 1 is presented by Mahankali et al. (2024a), in the implementation they introduce an intrinsic and extrinsic reward coefficient. In the experiments they choose intrinsic reward coefficient $\ll$ extrinsic reward coefficient. Thus, they are not directly optimizing equation 1 in the experiments. Furthermore, on page 3 the authors state that $\boldsymbol{z}$ "is resampled at the start of each trajectory". However, in the pseudocode on page 15 the authors note that $\boldsymbol{z}$ is resampled at the end of each trajectory or after a certain number of steps has passed. Since the authors implemented the latter, more general, case we adopt the latter approach for our experiments. The authors split the critic's head into two: one head predicts the intrinsic value function, the other head predicts the extrinsic value function. Figure 2 depicts how the action logits, intrinsic value, and extrinsic value for a given observation and $\boldsymbol{z}$ is calculated. Note the residual connection from the first element-wise addition to the second element-wise addition. There is no mention of this residual connection in the paper. However, it is present in the provided code for the Atari environment. To continue the example from above, in figure 1b the agent would prioritize to go left as this promises the greater reward. It is not hard to imagine that for higher dimensional environments, every time we start a new trajectory and correspondingly sample a new z, the agent will explore a different region of the state space and thus we can discover non-obvious paths to maximize the rewards. For the detailed pseudocode of this algorithm, see Appendix A (check link before submmission).

[2]In the pseudocode algorithm in line 19 on page 25, in the mathematical formulation, the authors state that the feature network is updated as a linear combination of the old feature network's weights and the policy network's weights. We contacted the authors, and they confirmed that it should read $\phi \leftarrow \tau \cdot V^\phi + (1 - \tau) \cdot \phi$ (i.e., the value network should be used, not the policy network). However, they state it would be interesting to explore which network should be used to update the feature network.

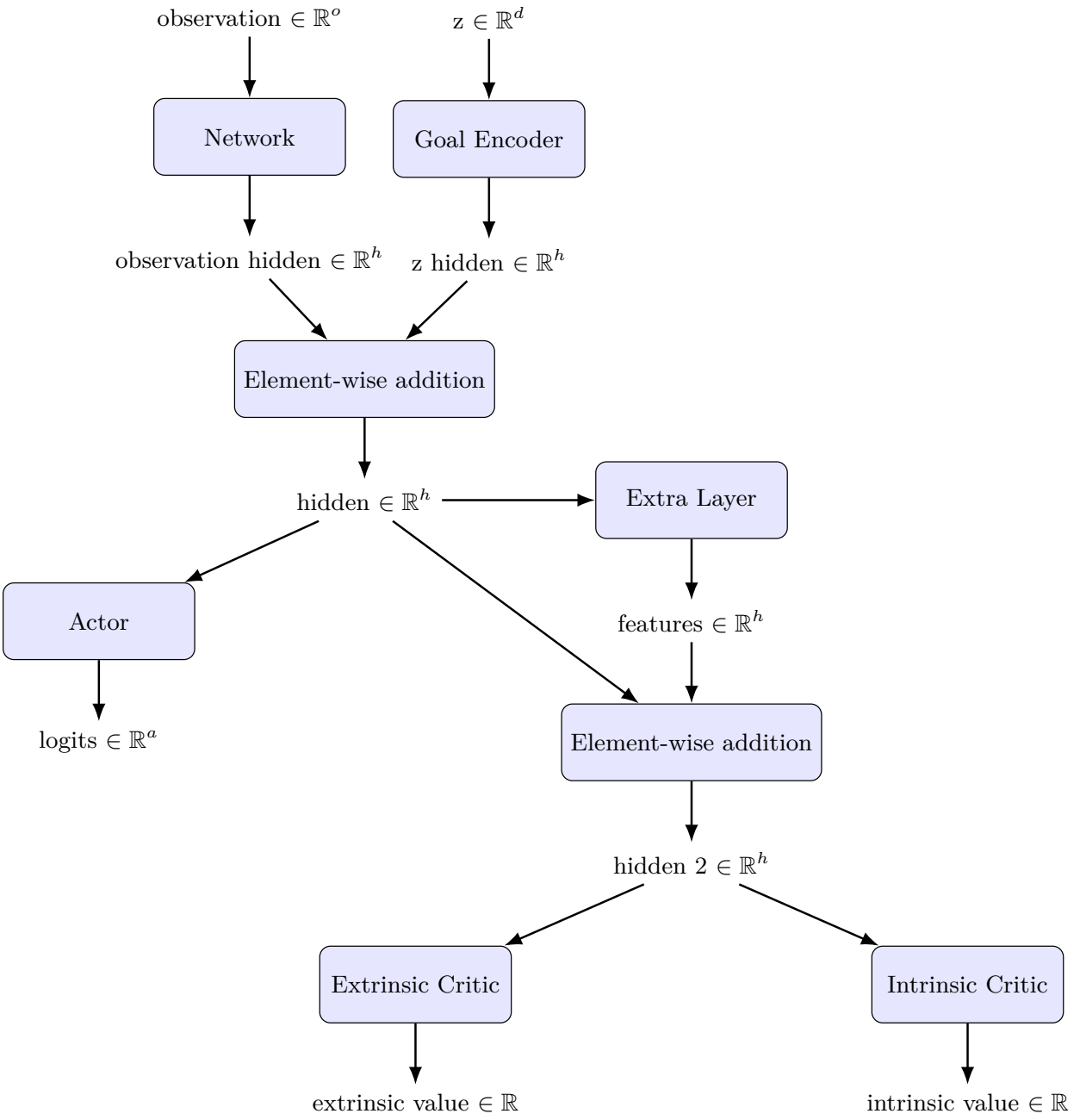


Figure 2: In RLE, the policy and critic networks are conditioned on the latent vector z. Mahankali et al. (2024a) split the critic's head into two: one head predicts the extrinsic value and the other head predicts the intrinsic value. In the vector dimension, o represents the dimension of a single observation, d represents the RLE feature dimension, and a is the cardinality of the action space.

1.2 Proximal Policy Optimization

The algorithm constituting the foundation of RLE is Proximal Policy Iteration (PPO), described in Schulman et al. (2017), which can itself be run to solve RL problems. It is an on-line policy gradient method where after running the policy $\boldsymbol{\pi}_{\theta_{\text{old}}}$ in parallel with N actors for $T << E$ timesteps, where E is the length of an episode, we compute the advantage estimator as proposed by Schulman et al. (2018) for each time index $t \in [0, T]$:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\,\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\,\delta_{T-1} \tag{2}$$

where we have $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$. Here, λ is the Generalized Advantage Estimation (GAE) parameter and $V(s)$ is a learned state-value function. Afterwards, for each t the surrogate objective is computed and the parameters θ are optimized e.g. with minibatch SGD or Adam (note that we optimize by maximizing the objective). This surrogate objective takes the form:

$$L_t^{\text{CLIP+VF+S}}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 S[\boldsymbol{\pi}_\theta](s_t) \right] \tag{3}$$

In equation 3, we have the VF coefficient c_1 and the entropy coefficient c_2, together with the (optional, to enhance exploration) entropy bonus $S[\boldsymbol{\pi}_\theta](s_t)$, which is calculated as the sum of $-p \log p$ over each of the action-probabilities p resulting from the policy distribution $\boldsymbol{\pi}_\theta$ for a state s_t, and the objectives:

$$L_t^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right] \tag{4}$$

$$L_t^{\text{VF}}(\theta) = \left(V_\theta(s_t) - V_t^{\text{targ}} \right)^2 \tag{5}$$

In equation 4 we denote the probability ratio as $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$, where $\pi_{\theta_{\text{old}}}$ is the policy before the update, and the clipping function is:

$$\text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right) = \begin{cases} 1-\epsilon, & r_t(\theta) < 1-\epsilon \\ r_t(\theta), & 1-\epsilon \leq r_t(\theta) \leq 1+\epsilon \\ 1+\epsilon, & r_t(\theta) > 1+\epsilon \end{cases} \tag{6}$$

Moreover in equation 5, V_t^{targ} represents some target state-value function that is to be obtained. Note that there exist also other objective functions mentioned in Schulman et al. (2017) that can substitute $L_t^{\text{CLIP}}(\theta)$ in equation 3. In equation 6, we make use of a hyperparameter $\epsilon \in [0, 1]$ and in combination with the minimizer from equation 4 we can prevent a single policy update to accidentally ruin the policy forever by sending the algorithm to a gradient region with a worse local extremum. The detailed pseudocode of this algorithm can be found in Appendix B (check link before submmission).

In the paper that we investigate, the authors Mahankali et al. (2024a) compare the empirical results of the RLE to PPO as the baseline, as well as two other exploration techniques, namely NoisyNet and RND, which will be briefly explained in the following sections.

1.3 NoisyNet

The central idea in a NoisyNet architecture brought forward by Fortunato et al. (2018) is to introduce random noise into the weights learned by the network (p inputs, q outputs), i.e. if we have a neural network whose output can be parametrized by a vector of weights $\boldsymbol{\theta}$ as $\mathbf{y} = f_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + \mathbf{b}$, then we can learn the set of parameter vectors $\zeta = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$ to compute $\mathbf{w} = (\boldsymbol{\mu}^w + \boldsymbol{\sigma}^w \odot \boldsymbol{\varepsilon}^w$ and $\mathbf{b} = (\boldsymbol{\mu}^b + \boldsymbol{\sigma}^b \odot \boldsymbol{\varepsilon}^b)$ with the following dimensionalities: $\boldsymbol{\mu}^w$, $\boldsymbol{\sigma}^w$, $\boldsymbol{\varepsilon}^w \in \mathbb{R}^{q \times p}$ implying $\mathbf{w} \in \mathbb{R}^{q \times p}$, and $\boldsymbol{\mu}^b$, $\boldsymbol{\sigma}^b$, $\boldsymbol{\varepsilon}^b \in \mathbb{R}^q$ resulting in $\mathbf{b} \in \mathbb{R}^q$. Note that $\mathbf{x} \in \mathbb{R}^p$, $\mathbf{y} \in \mathbb{R}^q$ and $\odot$ denotes element-wise multiplication.

We optimize the loss $\bar{L}(\zeta) = \mathbb{E}_\varepsilon[L(\boldsymbol{\theta})]$, i.e. the expectation of the loss $L(\boldsymbol{\theta})$ over the noise ε which has a mean of zero and fixed statistics, using gradient descent on the parameters ζ. The noise can be sampled either independently or in a factorized way from a Gaussian. The former method samples the noise $\varepsilon^w_{i,j} \in \boldsymbol{\varepsilon}^w$ and $\varepsilon^b_j \in \boldsymbol{\varepsilon}^b$ for every input to the network leading to a total of $pq+q$ random samples. Factorized sampling on the other hand greatly reduces computational complexity by factorizing $\varepsilon^w_{i,j} = f(\varepsilon_i)f(\varepsilon_j)$ and then $\varepsilon^b_j = f(\varepsilon_j)$ thereby lowering the number of required random samples to $p+q$, where $f : \mathbb{R} \to \mathbb{R}$. In Fortunato et al. (2018) the function $f(x) = \text{sign}(x)\sqrt{|x|}$ is used. After sampling the noise, in theory we compute the gradients $\nabla\bar{L}(\zeta) = \nabla\mathbb{E}_\varepsilon[L(\boldsymbol{\theta})] = \mathbb{E}_\varepsilon[\nabla_{\boldsymbol{\mu},\boldsymbol{\Sigma}} L(\boldsymbol{\mu}+\boldsymbol{\Sigma}\odot\boldsymbol{\varepsilon})]$, whereas in practice we approximate $\nabla\bar{L}(\zeta) \approx \nabla_{\boldsymbol{\mu},\boldsymbol{\Sigma}} L(\boldsymbol{\mu}+\boldsymbol{\Sigma}\odot\boldsymbol{\varepsilon})$ with the Monte Carlo method.

In the code of Mahankali et al. (2024a), a NoisyNet is used in conjunction with the A3C-algorithm, in particular the loss is computed from the action-value function is estimated using:

$$\hat{Q}_i = \sum_{j=i}^{k-1} \gamma^{j-i} r_{t+j} + \gamma^{k-i} V(x_{t+k}|\zeta, \varepsilon_i) \tag{7}$$

Moreover, for factorized networks, we initialize $\mu_{i,j} \sim \mathcal{U}\left[-\frac{1}{\sqrt{p}}, +\frac{1}{\sqrt{p}}\right]$ and $\sigma_{i,j} = \frac{\sigma_0}{\sqrt{p}}$ with p being the number of inputs to the corresponding linear layer, $\mathcal{U}$ is a uniform distribution over the specified interval, and σ_0 is a hyperparameter. The pseudocode for this variant given by Fortunato et al. (2018) can be found in appendix Appendix C Check link before submission.

Are we using independent or factorized sampling in the code? (I'm guessing factorized).

1.4 Random Network Distillation

In Random Network Distillation, as described by Burda et al. (2019), we compose the reward which the agent obtains as $r_t = e_t + i_t$ where e_t represents a sparse extrinsic (environment's) reward and i_t is the intrinsic reward for transitioning, i.e. the exploration bonus emanating from the transition at time step t. The latter measures the novelty of a state and should yield a higher value, the less frequently a state has been visited so far. In case of a finite state space we can use $i_t = \frac{1}{n_t}$ or $i_t = \frac{1}{\sqrt{s}}$ where $n_t(s)$ denotes the number of times state s has been visited until time step t. However, there exist alternatives, e.g. we could use state density based approaches to calculate an exploration bonus, which are described in the paper mentioned above. Moreover, they specify that in the paper the intrinsic reward is the prediction error of a randomly generated problem $i_t = \|\hat{f}(x|\theta) - f(x)\|^2$ involving a target network $f : \mathcal{O} \to \mathbb{R}^k$ (fixed and randomly initialized, maps an observation $\mathcal{O}$ to an embedding $\mathbb{R}^k$) to sample the problem as well as a predictor network $\hat{f} : \mathcal{O} \to \mathbb{R}^k$ which was trained on the agent's data by gradient descent to optimize i_t w.r.t. parameters $\theta_{\hat{f}}$.

Since the overall return can be composed as a sum of returns $R = R_E + R_I$ we can train two heads V_E and V_I to estimate the value function $V = V_E + V_I$ and in extension the advantages. In the end, a policy is trained with standard PPO using the advantage estimators. Note that the intrinsic rewards have to be normalized in order to be useful because otherwise we could not guarantee that they are on a consistent scale, which is done by dividing i_t by a running estimate of the standard deviations of the intrinsic return. Furthermore, the observation also has to be normalized as $x \leftarrow \text{clip}(\frac{x-\mu}{\sigma}, -5, 5)$. The normalization parameters can be initialized by letting a random agent briefly move in the environment before beginning the optimization. The pseudocode given by Burda et al. (2019) can be found in appendix Appendix D Check link before submission.

2 Scope of reproducibility

The main quantifiable claims made by Mahankali et al. (2024a) consist of the following:

1. In the abstract, Mahankali et al. (2024a) claim "[...] RLE exhibits higher overall scores across all of the tasks than other approaches, including action-noise and randomized value exploration." They refer to the tasks being the Atari, IssacGym and FourRoom environments. The benchmark approaches are standard PPO, NoisyNet and RND.

2. The first claim holds for "[...] both discrete and continuous control tasks." (Mahankali et al., 2024a)
3. Moreover, "[...] introducting randomness to rewards influences the policy to produce diverse behaviors" (Mahankali et al., 2024a) which manifests itself in an incentive to explore random (and thus in aggregate lager) parts of the state space.
4. For the discrete-action discrete-states FourRoom environment absent of a goal reward, the authors claim that PPO's state visitation centers around the initial room (top right) concluding that action-noise methods cannot conduct deep exploration whereas RLE, NoisyNet and RND well reach across the four rooms and thus perform deep exploration.
5. For the more challenging discrete-action continuous-states[3] Atari environment, the authors claim that, in the majority of games, RLE achieves a higher IQM of the human-normalized score with a probability of 67%, 72% and 71%, compared to PPO, NoisyNet and RND, respectively.[4]
6. In the continuous-action continuous-states IsaacGym environment[5], the authors claim that, with a probability of around 83% for PPO, ca. 66% for NoisyNet and roughly 65% for RND, RLE improves performance in all of their selected IsaacGym tasks.

Our findings regarding these claims will be presented in section 4 Check link before submission. Moreover, Mahankali et al. (2024a) performed ablation studies and claim to have found:

- RLE performance improves over PPO irrespective of the distribution used to sample the 8-dim random latent vector.
- RLE performance does not significantly improve over PPO as a dependence of the dimensionality of the random latent vector.
- RLE performance deteriorates if the policy and value networks are not conditioned on the random latent vector.
- RLE performance improves slightly if a learning feature network is used in comparison to when the feature network has constant weights when evaluated on the Atari games.
- RLE performance is insensitive to the feature extractor architecture.

Finally, the authors also conclude that: Unfortunately due to time constraints we are not able to test all of these claims, instead we will focus on the main ones.

Unfortunately due to time constraints we are not able to test all of these claims, instead we will focus on the main ones.

3 Methodology

The authors of the paper at hand published code in a GitHub repository[6]. This repository contains code for the Atari and IsaacGym[7] environments with the 4 algorithms (RLE, PPO, RND, NoisyNet). However, the authors did not provide any code for the FourRoom environment, so this environment was implemented by us using the gymnasium library introduced by Towers et al. (2024) together with our adaptation of the

[3] While the input to the action sampler is a discrete $84 \times 84 \times 4$ tensor representing the pixels it perceives, in practice Atari games are considered to be continuous-state environments due to the high-dimensional state space and visual complexity of each frame.

[4] The authors mention that for the Montezuma's Revenge game RLE underperforms likely due to RLE not factoring in bonus-based exploration.

[5] Please note that the IsaacGym environment is deprecated and we're using Isaac Lab (check this) instead, however we will refer to it as IsaacGym for continuity.

[6] https://github.com/Improbable-AI/random-latent-exploration

[7] Please note that the IsaacGym environment is deprecated and we are using IsaacLab instead, however we will refer to it as IsaacGym for reasons of continuity.

code for PPO, NoisyNet and RLE algorithms from the ATARI environment, while we adapted the RLE code from the ISAACGYM environment. A wrapper to record the state visitation counts per environment was also implemented.

Before we adapted RLE from the existing implementation for the ATARI environment, we implemented it based on the description in Mahankali et al. (2024a). However, the paper lacked a lot of details present in the ATARI code (e.g. split critic's head, residual connection) so that we decided to proceed with the adapted code to have consistent results over the different environments. In general, after coding the scripts, we uploaded the files to Google Colab and linked a Google Drive to save the models, as well as Weights & Biases

Alessia explains Atari stuff

Jonathan has the great pleasure of explaining now only Isaac but also the VMF stuff

`Explain your approach - did you use the author's code, or did you aim to re-implement the approach from the description in the paper? Summarize the resources (code, documentation, GPUs) that you used.`

3.1 Model descriptions

`Include a description of each model or algorithm used. Be sure to list the type of model, the number of parameters, and other relevant info (e.g. if it's pre-trained).`

3.2 Datasets

Since we are dealing with a reinforcement learning task, we did not use any dataset to train our models but rather four different environments which will explain in section 3.4 (check link before submission).

3.3 Hyperparameters

For the FOURROOM experiments the hyperparameters stated in Mahankali et al. (2024a) were used. However, Mahankali et al. (2024a) did not provide values for all available hyperparameters. For the missing hyperparameters we used the following values, that seemed reasonable to us:

Table 1: Hyperparameters not stated by Mahankali et al. (2024a) for the FOURROOM experiments.

Hyperparameter	**Value**
Extrinsic Reward Coefficient	1
Observation Normalization Iterations	1
Discount Rate	0.99
Intrinsic Discount Rate	0.99
Feature Network Update Rate τ	0.005
Learning Rate	0.001

`Describe how the hyperparameter values were set and what was the source for their value (e.g. paper, code, or your guess). If there was a hyperparameter search done, be sure to include the range of hyperparameters searched over, the method used to search (e.g. manual search, random search, grid search, etc.), and the best hyperparameters found. Include the number of total experiments (e.g. hyperparameter trials). You can also include all results from that search (not just the best-found results).`

3.4 Experimental setup and code

The code for our implementations can be found in GitHub[8]. There are three types of environments in use on which we run our RL algorithms which are depicted in figure 3.

Atari represents "discrete action space deep RL benchmark". The human normalized scores are computed according to formula: Agent57: Outperforming the Atari Human Benchmark and the human scores are taking from

Do we need to explain the environments as well?

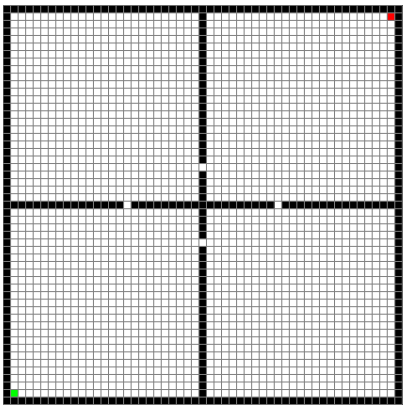
Figure 3a shows the FourRoom environment where we have $50 \times 50 + 4$ states[9] which are divided into four rooms of size 25×25 and 4 holes in the walls located at positions 10 horizontally and 5 vertically, starting the count at 1 from the inner corners. Exploring the state space, beginning from the initial state marked in red, becomes a deep exploration task due to the holes being only 1 "pixel" wide. However, if we want to incentivize the agent to find a reward, then we can put it at the goal position marked in green. At every step, the agent can move one step vertically or horizontally, but if it would run into a wall it chooses to remain in place. Mahankali et al. (2024a) refer to Sutton et al. (1999) when introducing the FourRoom environment. Sutton et al. (1999) use action stochasticity. We asked Mahankali et al. (2024a) whether they also used action stochasticity, and they stated that they did not use action stochasticity in their experiments. Mahankali et al. (2024a) claims that RLE shows good exploration behavior based on a single heatmap of state visitation counts. To assess whether this result is expected or exceptional we trained each algorithm 20 times. Since it is not reasonable to compare 20 state visitation count heatmaps we used the Shannon entropy, introduced by Shannon (1948), of the state visitation counts as a proxy for how well an agent explored the environment. With increasing entropy the distribution of the state visitation counts becomes more uniform. Thus, the higher the entropy the better the explorative behavior. Furthermore, it was explored if the intrinsic value function really guides the generation of diverse trajectories. To assess this, the intrinsic value function for four latent vectors z and four trajectories generated using the same latent vector z were saved at the end of each experiment. We also tested to what degree RLE is indifferent to the distribution of the latent vector. To test this, we ran RLE with different latent vector distributions. The following distributions were considered: standard normal distribution, standard uniform distribution, Von-Mises distribution with $\mu = 0$ and $\kappa = 0.3$, and the exponential distribution with $\lambda = 0.3$. We ran 20 experiments for each latent vector distribution and recorded the state visitation entropy. The same hyperparameters were used as before. Note that all latent vectors were normalized using the L2-norm, so that all latent vectors are on a unit sphere around the origin. This ensures that all intrinsic rewards are in the interval [-1,1]. Mahankali et al. (2024a) performed a similar ablation study for the Atari environment. However, they did not normalize the latent vectors for all distributions. They only applied normalization for the normal distribution which they then denoted as "Sphere", that they compare to the unnormalized uniform distribution and the unnormalized normal distribution. We normalized all distributions because the "extrinsic reward coefficient" and the "intrinsic reward coefficient" hyperparameters already control the weighting between the intrinsic and extrinsic rewards, so we wanted the intrinsic rewards to be in the interval [-1,1].

Moreover we have an example of the Atari environments in figure 3b, namely the Alien-v5 game. In this environment, the agent perceives the 4 most recent 84×84 grayscale frames. In the IsaacGym environment on the other hand, <finish the environment description later>.

`Include a description of how the experiments were set up that's clear enough a reader could replicate the setup. Include a description of the specific measure used to evaluate the experiments (e.g. accuracy, precision@K, BLEU score, etc.). Provide a link to your code or notebook (if available). Add in this section (or in the following) any reference to cloud-based providers you`

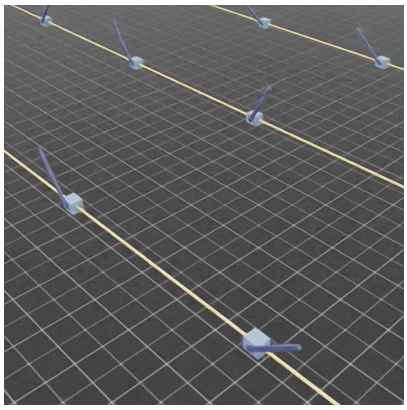
[8] https://github.com/jonupp/adv_topics_ml_repl_chal Change github

[9] In Mahankali et al. (2024a), the authors claim that there are 50×50 states. We contacted the authors and they confirmed that it should read $50 \times 50 + 4$ states.



(a) The FourRoom environment is a grid world, limited by walls, with the initial state is on the top right and the goal state where the agent receives a reward (optional) is on the bottom left.

(b) The Alien-v5 environment is an example of the Atari games consisting of a maze filled with 'eggs' to destroy while being hunted by aliens. A flamethrower or occasional power-up may be used to scare the aliens.



(c) The CartPole environment is an example of the IsaacGym environment.
What does the agent actually perceive?

Figure 3: Examples of the environments in use.

3.5 Computational requirements

The code for the FourRoom experiments was exclusively run on the CPU. For Atari, we experimented with the CPU, T4 GPU and TPU v2-8 in Google Colab. Weights & Biases (wandb) was used for easier runtime visualization of the scores and other measures.

We would like to note, however, that Mahankali et al. (2024a) had available the HPC resources of the MIT Supercloud and the Lincoln Laboratory Supercomputing Center. Within the framework of this course, we did not have the equivalent computing power to replicate all experiments in their original form and had to restrain ourselves to a mere few of them.

```
Include a description of the hardware used, such as the GPU or CPU the experiments were
run on.  For each model, include a measure of the average runtime (e.g.  average time
to predict labels for a given validation set with a particular batch size).  For each
experiment, include the total computational requirements (e.g.  the total GPU hours
spent).
Note:  You'll likely have to record this as you run your experiments, so it's better to
think about it ahead of time.
```

4 Results

```
Start with a high-level overview of your results.  Do your results support the main
claims of the original paper?  Keep this section as factual and precise as possible,
and reserve your judgment and discussion points for the next "Discussion" section.
```

4.1 Results reproducing original paper

```
For each experiment, say (1) which claim in Section 2 it supports, and (2) if it
successfully reproduced the associated experiment in the original paper.      For example,
an experiment training and evaluating a model on a dataset may support a claim that that
model outperforms some baseline. Logically group related results into subsections.
```

4.1.1 Result 1

4.1.2 Result 2

4.2 Results beyond original paper

Often papers don't include enough information to fully specify their experiments, so some additional experimentation may be necessary. For example, it might be the case that batch size was not specified, and so different batch sizes need to be evaluated to reproduce the original results. Include the results of any additional experiments here. Note: this won't be necessary for all reproductions.

4.2.1 Additional Result 1

4.2.2 Additional Result 2

5 Discussion

Give your judgment on if your experimental results support the claims of the paper. Discuss the strengths and weaknesses of your approach - perhaps you didn't have time to run all the experiments, or perhaps you did additional experiments that further strengthened the claims in the paper.

5.1 What was easy

Atari code was ready to run.

Give your judgment of what was easy to reproduce. Perhaps the author's code is clearly written and easy to run, so it was easy to verify the majority of original claims. Or, the explanation in the paper was really easy to follow and put into code.

Be careful not to give sweeping generalizations. Something that is easy for you might be difficult for others. Put what was easy in context and explain why it was easy (e.g. code had extensive API documentation and a lot of examples that matched experiments in papers).

5.2 What was difficult

- Isaac deprecated
- FourRoom env missing
- Inconsistencies in the paper and lack of descriptions
- Hardware constraints‘’
- Discrepancy between paper and code

List part of the reproduction study that took more time than you anticipated or you felt was difficult.

Be careful to put your discussion in context. For example, don't say "The maths was difficult to follow", say "The math requires advanced knowledge of calculus to follow".

5.3 Communication with original authors

Document the extent of (or lack of) communication with the original authors. To make sure the reproducibility report is a fair assessment of the original research we recommend getting in touch with the original authors. We advise you to contact them through their email displayed in the paper and by asking specific questions.

6 Conclusion

Try to summarize the achievements of your project and its limits, suggesting (when appropriate) possible extensions and future works.

Member contributions

We send him an email got answer.

Include a section specifying how you organized the work within the group and clearly describe the contributions of each member. Make sure to properly balance the workload among the members.

References

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1lJJnR5Ym.

Meire Fortunato, Mohammad G. Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. *International Conference on Machine Learning*, 2018. doi: 10.48550/arXiv.1706.10295.

Srinath V. Mahankali, Zhang-Wei Hong, Ayush Sekhari, Alexander Rakhlin, and Pulkit Agrawal. Random latent exploration for deep reinforcement learning (new version). *Forty-first International Conference on Machine Learning*, 2024a. doi: 10.48550/arXiv.2407.13755. URL https://srinathm1359.github.io/random-latent-exploration/.

Srinath V. Mahankali, Zhang-Wei Hong, Ayush Sekhari, Alexander Rakhlin, and Pulkit Agrawal. Random latent exploration for deep reinforcement learning (old version). 2024b. URL https://openreview.net/forum?id=Y9qzwNlKVU.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017. doi: 10.48550/arXiv.1707.06347.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018. URL https://arxiv.org/abs/1506.02438.

C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3): 379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999. URL https://www.sciencedirect.com/science/article/pii/S0004370299000521.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024. URL https://arxiv.org/abs/2407.17032.

A Pseudocode – Random Latent Exploration

For the RLE algorithm, the detailed pseudocode according to Mahankali et al. (2024a) is as follows:

RLE:	Detailed Pseudocode
1:	**Input:** Latent distribution ($P_{\mathbf{z}}$), number of parallel workers (N), number of steps per update (T), number of steps per sampling (S), and feature network update rate (τ).
2:	Randomly initialize a feature network (ϕ) with the same backbone architecture as the policy and value networks.
3:	Initialize running mean $\boldsymbol{\mu} = \mathbf{0}$ and standard deviation $\boldsymbol{\sigma} = \mathbf{1}$ estimates of $\boldsymbol{\phi}(s)$ over the state space.
4:	Sample an initial latent vector $\mathbf{z} \sim P_{\mathbf{z}}$ for each parallel worker.
5:	**Repeat** $(^1)$
6:	| Sample initial state s_0.
7:	| **For** $t = 0, ..., T$ **do** $(^2)$
8:	| | Take action $a_t \sim \boldsymbol{\pi}(.\|s_t, \mathbf{z})$ and transition to s_{t+1}.
9:	| | Compute feature $\mathbf{f}(s_{t+1}) = \frac{\boldsymbol{\phi}(s_{t+1}) - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$.
10:	| | Compute random reward $F(s_{t+1}, \mathbf{z}) = \frac{\mathbf{f}(s_{t+1})}{\|\mathbf{f}(s_{t+1})\|} \cdot \mathbf{z}$.
11:	| | Receive reward $r_t = R(s_t, a_t) + F(s_{t+1}, \mathbf{z})$.
12:	| | **For** $i = 0, 1, ..., N-1$ **do** $(^3)$
13:	| | | **If** worker i terminated **or** S timesteps passed without resampling, **then** $(^4)$
14:	| | | | Resample sample $\mathbf{z} \sim P_{\mathbf{z}}$ for worker i.
15:	| | | $(^4)$
16:	| | $(^3)$
17:	| $(^2)$
18:	| Update policy network $\boldsymbol{\pi}$ and value network $V_{\boldsymbol{\pi}}$ with the collected trajectory $(\mathbf{z}, s_0, a_0, r_0, s_1, ..., s_T)$.
19:	| Update feature network $\boldsymbol{\phi}$ using the value network's parameters $\boldsymbol{\phi} \leftarrow \tau \cdot V^{\boldsymbol{\pi}} + (1-\tau) \cdot \boldsymbol{\phi}$.
20:	| Update $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ using the batch of collected experience.
21:	$(^1)$ **until convergence**

B Pseudocode – Proximal Policy Optimization

For the PPO algorithm, the detailed pseudocode according to Schulman et al. (2017) is as follows:

PPO:	Detailed Pseudocode, Actor-Critic Setup
1:	**Input:** Number of iterations (I), number of actors (N), number of timesteps per iteration (T), number of epochs (K), minibatch size ($M \leq NT$), learned state-value function implicitly in the advantage estimator ($V(s)$), target value (V_t^{targ}) implicitly in the objective L.
2:	**For** $i = 1, ..., I$ **do** $(^1)$
3:	| **For** $n = 1, ..., N$ **do** $(^2)$
4:	| | Run policy $\boldsymbol{\pi}_{\theta_{\text{old}}}$ in environment for T timesteps.
5:	| | Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$.
6:	| $(^2)$
7:	| Optimize objective L w.r.t. θ, for given K and M, using e.g. minibatch SGD or Adam.
8:	| Update parameters $\theta_{\text{old}} \leftarrow \theta$.
9:	$(^1)$

C Pseudocode – NoisyNet

For the NoisyNet + A3C algorithm, the detailed pseudocode according to Fortunato et al. (2018) is as follows:

NoisyNet: Detailed Pseudocode, for each actor-learner thread

Input: Global shared parameters (ζ_π, ζ_V), global shared counter (T), and maximal time $(T_{\max})$.
Input (each thread): Thread-specific parameters (ζ'_π, ζ'_V), set of random variables (ε), thread-specific counter (t), and roll-out size $(t_{\max})$.
Output: Policy $\pi(\cdot|\zeta_\pi, \varepsilon)$ and value function $V(\cdot|\zeta_V, \varepsilon)$.
$t \leftarrow 1$
Repeat $(^1)$
  Reset cumulative gradients: $d\zeta_\pi \leftarrow 0, d\zeta_V \leftarrow 0$.
  Synchronize thread-specific parameters: $\zeta'_\pi \leftarrow \zeta_\pi, \zeta'_V \leftarrow \zeta_V$.
  Set counter $c \leftarrow 0$.
  Get state x_t from environment.
  Sample noise $\xi \sim \varepsilon$.
  Create lists for rewards $r \leftarrow []$, actions $a \leftarrow []$ and states $x \leftarrow []$.
  Repeat $(^2)$
    Choose action $a_t \leftarrow \pi(\cdot|x_t, \zeta'_{pi}, \zeta'_V)$.
    $a[-1] \leftarrow a_t$
    Obtain reward r_t and new state x_{t+1}.
    $r[-1] \leftarrow r_t$, $x[-1] \leftarrow x_t$, $t \leftarrow t+1$, $T \leftarrow T+1$, $c \leftarrow c+1$
  $(^2)$ **until** x_t **is terminal or** $c > t_{\max}$
  If x_t is terminal, **then** $(^3)$
    $Q \leftarrow 0$
  else
    $Q \leftarrow V\left(x_t|\zeta'_V, \xi\right)$
  $(^3)$
  For $i = c-1, ..., 0$ **do** $(^4)$
    $Q \leftarrow r\left[i\right] + \gamma Q$
    $d\zeta_\pi \leftarrow d\zeta_\pi \nabla_{\zeta'_\pi} \log\left(\pi(a[i]|x[i], \zeta'_\pi, \xi)\right)[Q - V(x[i]|\zeta'_V, \xi)]$
    $d\zeta_V \leftarrow d\zeta_V \nabla_{\zeta'_V} \left[Q - V\left(x[i]|\zeta'_V, \xi\right)\right]^2$
  $(^4)$
  Update asynchronously parameter $\zeta_\pi \leftarrow \zeta_\pi + \alpha_\pi d\zeta_\pi$.
  Update asynchronously parameter $\zeta_V \leftarrow \zeta_V - \alpha_V d\zeta_V$.
$(^1)$ **until** $T > T_{\max}$

D Pseudocode — Random Network Distillation

For the RND algorithm, the detailed pseudocode according to Burda et al. (2019) is as follows:

RND: Detailed Pseudocode

Input: Number of rollouts (N), Number of optimization steps (N_{opt}), and length of initial steps for initializing observation normalization (M).
$t \leftarrow 0$
Sample state $s_0 \sim p_0(s_0)$.
For $m = 1, ..., M$ **do** **(1)**
  Sample action $a_t \sim \mathcal{U}(a_t)$. Are we sampling uniformly from all available actions at time t?
  Sample state $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$.
  Update observation normalization parameters using s_{t+1}.
  $t \leftarrow t + 1$
(1)
For $i = 1, ..., N$ **do** **(2)**
  For $j = 1, ..., K$ **do** **(3)**
    Sample action $a_t \sim \pi(a_t|s_t)$.
    Sample state $s_{t+1}, e_t \sim p(s_{t+1}, e_t|s_t, a_t)$.
    Calculate intrinsic reward $i_t = \|\hat{f}(s_{t+1}) - f(s_{t+1})\|^2$.
    Add $s_t, s_{t+1}, a_t, e_t, i_t$ to optimization batch B_i.
    Update running estimate of reward standard deviation using i_t.
    $t \leftarrow t + 1$
  (3)
  Normalize the intrinsic rewards contained in B_i.
  Calculate returns $R_{I,i}$ and advantages $A_{I,i}$ for intrinsic reward.
  Calculate returns $R_{E,i}$ and advantages $A_{E,i}$ for extrinsic reward.
  Calculate combined advantages $A_i = A_{I,i} + A_{E,i}$.
  Update observation normalization parameters using B_i.
  For $j = 1, ..., N_{\text{opt}}$ **do** **(4)**
    Optimize θ_π w.r.t. PPO loss on batch B_i, R_i, A_i using Adam.
    Optimize $\theta_{\hat{f}}$ w.r.t. distillation loss on B_i using Adam.
  (4)
(2)

E Citations, figures, and tables

Citations When the authors or the publication are included in the sentence, the citation should not be in parenthesis, using `\citet{}` (as in "See (CITATION REMOVED) for more information."). Otherwise, the citation should be in parenthesis using `\citep{}` (as in "Deep learning shows promise to make progress towards AI (CITATION REMOVED).").

Figures All artwork must be neat, clean, and legible. The figure number and caption always appear after the figure. Place one line space before the figure caption, and one line space after the figure. The figure caption is lowercase (except for the first word and proper nouns). Make sure the figure caption does not get separated from the figure. Leave sufficient space to avoid splitting the figure and figure caption.

Tables All tables must be centered, neat, clean and legible. The table number and title always appear before the table. See Table 2. Place one line space before the table title, one line space after the table title, and one line space after the table. The table title must be lowercase (except for the first word and proper nouns).

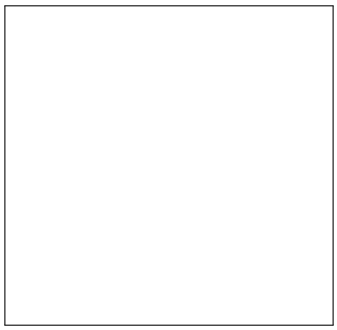


Figure 4: Sample figure caption.

Table 2: Sample table title

PART	DESCRIPTION
Dendrite	Input terminal
Axon	Output terminal
Soma	Cell body (contains cell nucleus)

F Math Notation

In an attempt to encourage standardized notation, we have included the notation file from the textbook, *Deep Learning* (CITATION REMOVED) available at `https://github.com/goodfeli/dlbook_notation/`. Use of this style is not required and can be disabled by commenting out `math_commands.tex`.

Numbers and Arrays

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
$\mathbf{A}$	A tensor
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
$\boldsymbol{e}^{(i)}$	Standard basis vector $[0, \dots, 0, 1, 0, \dots, 0]$ with a 1 at position i
$\text{diag}(\boldsymbol{a})$	A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable

Sets and Graphs

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \dots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\mathbb{A} \backslash \mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$
$\mathcal{G}$	A graph
$Pa_{\mathcal{G}}(\mathrm{x}_i)$	The parents of x_i in $\mathcal{G}$

Indexing

a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
a_{-i}	All elements of vector $\boldsymbol{a}$ except for element i
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$A_{i,j,k}$	Element (i, j, k) of a 3-D tensor $\mathbf{A}$
$\mathbf{A}_{:,:,i}$	2-D slice of a 3-D tensor
a_i	Element i of the random vector $\mathbf{a}$

Calculus

$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matrix derivatives of y with respect to $\boldsymbol{X}$
$\nabla_{\mathbf{X}} y$	Tensor containing derivatives of y with respect to $\mathbf{X}$
$\frac{\partial f}{\partial \boldsymbol{x}}$	Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \to \mathbb{R}^m$
$\nabla^2_{\boldsymbol{x}} f(\boldsymbol{x})$ or $\boldsymbol{H}(f)(\boldsymbol{x})$	The Hessian matrix of f at input point $\boldsymbol{x}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral over the entire domain of $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral with respect to $\boldsymbol{x}$ over the set $\mathbb{S}$

Probability and Information Theory

$P(\mathrm{a})$	A probability distribution over a discrete variable
$p(\mathrm{a})$	A probability distribution over a continuous variable, or over a variable whose type has not been specified
$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathbb{E}_{\mathrm{x} \sim P}[f(x)]$ or $\mathbb{E} f(x)$	Expectation of $f(x)$ with respect to $P(\mathrm{x})$
$\mathrm{Var}(f(x))$	Variance of $f(x)$ under $P(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Covariance of $f(x)$ and $g(x)$ under $P(\mathrm{x})$
$H(\mathrm{x})$	Shannon entropy of the random variable x
$D_{\mathrm{KL}}(P \Vert Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Functions

$f : \mathbb{A} \to \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f \circ g$	Composition of the functions f and g
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$. (Sometimes we write $f(\boldsymbol{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation)
$\log x$	Natural logarithm of x
$\sigma(x)$	Logistic sigmoid, $\dfrac{1}{1+\exp(-x)}$
$\zeta(x)$	Softplus, $\log(1+\exp(x))$
$\|\|\boldsymbol{x}\|\|_p$	L^p norm of $\boldsymbol{x}$
$\|\|\boldsymbol{x}\|\|$	L^2 norm of $\boldsymbol{x}$
x^+	Positive part of x, i.e., $\max(0, x)$
$\mathbf{1}_{\text{condition}}$	is 1 if the condition is true, 0 otherwise