Analysis of the Information Plane for Deep Reinforcement Learning Using Proximal Policy Optimization

Arthur F. S. Fernandes
School of Electrical and Computer Engineering
Universidade Estadual de Campinas
Campinas, Brazil
a257318@dac.unicamp.br

Denis G. Fantinato
School of Electrical and Computer Engineering
Universidade Estadual de Campinas
Campinas, Brazil
denisf@unicamp.br

***Abstract*—The Information Bottleneck (IB) principle has been explored in Deep Reinforcement Learning (DRL) approaches for control of the flow of information within the deep neural networks (DNNs). The application of IB in DNNs exhibited a considerable improvement in generalization and in the reduction of overfitting, allowing greater robustness against environmental variations. Regarding the supervised paradigm, the IB framework, along with its extension to the Information Plane, is also used for analysis and/or evaluation of the training efficiency, compression of information, performance in regression/classification, complexity of the DNN architecture, among others. However, for DRL, the analysis through the IB perspective is still incipient. In that sense, in this work, we analyze the use of the IB and the Information Plane in DRL, using the Proximal Policy Optimization (PPO) algorithm in the CartPole environment. The results show that the learning process in DRL exhibits phases of information compression and expansion within the network's layers, mirroring certain observations from supervised learning. Interestingly, different layers exhibit varying information flow patterns, with complex, not well-behaved trajectories in the Information Plane, suggesting a layer-specific adaptation to the task.**

***Index Terms*—Deep Reinforcement Learning; Proximal Policy Optimization; Information Bottleneck; Information Plane.**

I. Introduction

Reinforcement Learning (RL) is a powerful paradigm within Machine Learning (ML) that enables agents to learn optimal decision-making strategies through an iterative process of trial and error [1]. Unlike supervised learning, which relies on labeled data, RL agents learn by interacting with an environment and receiving feedback in the form of rewards or penalties for their actions [2]. This dynamic interaction fosters the development of sophisticated policies that guide the agent's behavior towards maximizing cumulative rewards. Its capability to learn from unlabeled data has attracted great attention in literature, finding applications in a wide range of scenarios, such as motion control and a set of decision making tasks [3], [4].

Traditional RL methods, however, often struggle with the complexities of real-world environments, which typically involve high-dimensional observations and intricate decision spaces [1]. In this context, Deep Learning (DL) methods emerge as a crucial tool, being excellent at extracting meaningful representations from raw data through the information processing by multiple layers of simple functions in a deep neural network (DNN) [5]. Each layer transforms the input data into a higher-level representation, gradually capturing increasingly complex patterns and features [5]. The combination of both approaches, pioneered by [6], is usually referred to as Deep Reinforcement Learning (DRL) [2].

DRL leverages the representational power of deep neural networks to address the challenges of complex environments and high-dimensional observations, enabling agents to learn sophisticated policies and achieving remarkable results in complex domains such as game playing [7], [8], robot motion control [3] and (partial) training of large language models [9].

The use of DNNs on the other hand, carries its own idiosyncrasies. One of the main objectives of DL is to achieve an efficient representation through the processing architecture, which should maximize the flux of relevant information for the current task, while minimizing irrelevant aspects of the input and noise [5]. This notion is suitably represented by the Information Bottleneck (IB) principle, which provides a rich theoretical framework for understanding the information flow and guiding the learning process of efficient representations [10].

Basically, the IB approach is based on the information theoretic principle to measure relevant information that an input contains about an output by means of the mutual information metric [10]. More interestingly, in DNNs, IB can also be applied to hidden layers or units for assessing the information flow through the network [11]. In the supervised approach, the IB visualization by means of the Information Plane (IP) [12] has brought insightful ideas and a deeper understanding on the learning process, allowing, besides the analysis of the network efficiency, the identification of distinct stages during learning [11].

In RL, the IB principle has been used to improve generalization by controlling the flow of information between the agent's observations, internal representations, and actions [13]–[15].

This is achieved by minimizing the mutual information between the state and the representation (to encourage compression) while maximizing the mutual information between the representation and the action or value function (to ensure task-relevant information is preserved). By applying IB, DRL methods can reduce overfitting to noisy or redundant features, enhance robustness to environmental variations and improve performance in unknown or partially observable environments. This approach has been particularly useful in high-dimensional state spaces, where learning efficient and generalizable representations is critical [14]. However, the study of the IP, including an analysis of the learning trajectory, is, to the best of our knowledge, still lacking in literature.

Since the application of the IB principle and the IP has shown promising results for DL in the supervised paradigms, in this work, we propose to perform a study of its applicability within the DRL framework. By applying the IP and the IB principles to guide the learning process in DRL agents, it becomes possible to enhance their sample efficiency, improve generalization, and ultimately lead to more robust and data-efficient algorithms.

II. Deep Reinforcement Learning

Reinforcement learning involves training an agent to select the most appropriate action in a given state to maximize a cumulative reward over a sequence of states [1].

In this context, an agent is defined as an entity capable of observing the environment through sensors and also modifying this environment through its actions, as illustrated in Figure 1. The environment is the total space where the agent is located. The agent, in turn, represents its observations of the environment at a given time step k through a state s from the set of states $\mathcal{S}$. From this, the agent will interfere with the environment with an action a from the set of actions $\mathcal{A}$. Consequently, the agent will be taken to a new state s' and receive the associated reward r from the set of rewards $\mathcal{R}$ [16].

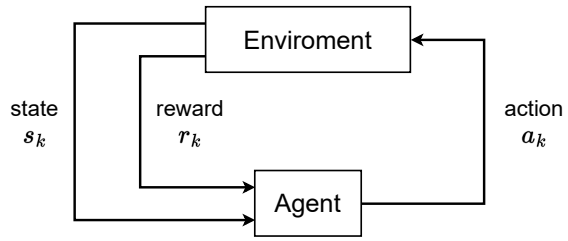

Fig. 1. Agent-environment interaction in a Markov decision process.

The goal of reinforcement learning is to obtain an agent that executes a certain "intelligent behavior", which, in practical terms, is explored through a more functional concept: the reward function [1]. After executing an action in a given state, the agent receives a value from the reward function, i.e., a mapping $R : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$, where $\mathcal{R} \subseteq \mathbb{R}$. The policy defines the agent's behavior in the environment: given a state s, a policy π returns an action a, making the mapping $\pi : \mathcal{S} \to \mathcal{A}$. It is worth noting that the policy must define an action for each state of the environment and it can be established after training the agent. The policy can also be stochastic, represented by a probability distribution $\pi(a|s)$ [1].

The learning process in RL involves finding an efficient policy that maximizes the expected cumulative reward. There is a variety of methods for addressing the challenge of discovering optimal policies in RL, in which we mention the REINFORCE algorithm [17], Actor-Critic (A2C, A3C and SAC) [18], Natural Policy Gradient (NPG) [19], among others. In particular, Actor-Critic methods present a compelling approach, combining the strengths of policy-based and value-based approaches [2], as described below.

A. Actor-Critic Methods

The Actor-critic methods provide an efficient framework for learning such policies by simultaneously learning a policy (actor) and an estimate of the value function (critic) [1]. The actor learns to select actions that maximize expected rewards, while the critic provides an evaluation of the expected future rewards for different states and actions, guiding the actor's learning process.

These methods parametrize the policy in function of $\boldsymbol{\theta}$, i.e. $\pi_{\boldsymbol{\theta}}(s) = \Pr[a|s, \boldsymbol{\theta}]$ for the Actor, and also use a learning model $\boldsymbol{\phi}$ to approximate the value $\hat{V}_{\boldsymbol{\phi}}(s)$ or the action-value function $\hat{Q}_{\boldsymbol{\phi}}(s, a)$ for the critic, being $\boldsymbol{\phi}$ the weights of the value function approximation function. Generally, neural networks are used to model the policy and value function, so that $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ represent their weights.

Considering an actor-critic, the weight update of the actor is described by:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \left(R(s,a) + \gamma \hat{V}_{\boldsymbol{\phi}}(s') - \hat{V}_{\boldsymbol{\phi}}(s) \right) \nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(a|s) \quad (1)$$

where α is the learning rate and $\ln(\cdot)$ is the natural logarithm. For the Critic, the weights are updated as:

$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \beta \left(R(s,a) + \gamma \hat{V}_{\boldsymbol{\phi}}(s') - \hat{V}_{\boldsymbol{\theta}}(s) \right) \nabla_{\boldsymbol{\phi}} \hat{V}_{\boldsymbol{\phi}}(s') \quad (2)$$

where β is the learning rate.

When neural networks are used to model the policy, their weights are updated through backpropagation [5]. For the policy network (actor), backpropagation is obtained based on the entropy loss:

$$\mathcal{L}_H = \mathbb{E}\left[-\sum_a \pi_{\boldsymbol{\theta}}(a|s) \ln \pi_{\boldsymbol{\theta}}(a|s) \right], \quad (3)$$

while, for the value network (critic), the gradient is $\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\boldsymbol{\phi}}$, with loss function:

$$\mathcal{L}_{\boldsymbol{\phi}} = \mathbb{E}\left[\left(R(s,a) + \gamma \hat{V}_{\boldsymbol{\phi}}(s') - \hat{V}_{\boldsymbol{\theta}}(s) \right)^2 \right], \quad (4)$$

being $\mathbb{E}[\cdot]$ the statistical expectation operator.

B. Proximal Policy Optimization

The current state-of-the-art in reinforcement learning developed based on the Actor-Critic methods and TRPO (Trust Region Policy Optimization) [20].

Proximal Policy Optimization (PPO) [20] uses a delayed version/estimation of the policy $\pi_{\boldsymbol{\theta}_{old}}(a|s)$ to perform a transformation based on importance sampling in order to estimate the reward gain that the new policy has when compared to the old one. This approach allows the samples reuse, contributing for a faster convergence of the algorithm. Mathematically, PPO aims at the maximization of

$$\mathcal{L}^{CLI}(\boldsymbol{\theta}) = \mathbb{E}\left[\frac{\pi_{\boldsymbol{\theta}}(a|s)}{\pi_{\boldsymbol{\theta}_{old}}(a|s)}\hat{A}_k\right] = \mathbb{E}\left[r_k(\boldsymbol{\theta})\hat{A}_k\right], \quad (5)$$

where $\hat{A}_k$ is the Advantage function defined as $\hat{A}(s_k, a_k) = \hat{Q}(s_k, a_k) - \hat{V}(s_k)$.

From Equation (5), PPO adds a clip $\mathcal{C}(\boldsymbol{\theta}) = \text{clip}(r_k(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\hat{A}_k$ and pairs it with a min operator as show in (6), where ϵ is a hyperparameter that varies in the interval $[0, 1]$. This limits the contribution that an advantageous action has on the agent, but allows unlimited penalty when the action is disadvantageous:

$$\mathcal{L}^{CLIP}(\boldsymbol{\theta}) = \mathbb{E}\left[\min\left(r_k(\boldsymbol{\theta})\hat{A}_k, \mathcal{C}(\boldsymbol{\theta})\right)\right]. \quad (6)$$

The goal is to maintain a good balance between stability and learning speed: advantageous actions are encouraged, but without radically changing the previous trajectory; while disastrous actions are highly penalized.

Considering the training of the Actor and Critic models, the total loss in PPO also makes use of the entropy loss (Eq. (3)) and value loss (Eq. (4)) is given by:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathcal{L}^{CLIP}(\boldsymbol{\theta}) + c_1\mathcal{L}_{\boldsymbol{\phi}} + c_2\mathcal{L}_H, \quad (7)$$

where c_1 and c_2 are weights selected to establish a balance among the terms.

Once selected the state-of-the-art algorithm, we shall analyze it under the IB and IP perspective.

III. Information Bottleneck and Information Plane

The IB method is an information theoretic principle for the analysis and/or control of the information flow within a deep network, by measuring relevant information that an input random variable $X \in \mathcal{X}$ contains about an output random variable $Y \in \mathcal{Y}$ [11]. In this case, the amount of information about X in Y is measured through the Mutual Information (MI):

$$I(X, Y) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p_{(X,Y)}(x, y)\log\frac{p_{(X,Y)}(x, y)}{p_X(x)p_Y(y)}, \quad (8)$$

where $p_X(x)$ and $p_Y(y)$ are the probability mass function (PMF) of discrete variables X and Y, respectively, and $p_{(X,Y)}(x, y)$ is the joint probability mass function of X and Y.

In the context of DNNs, the IB can be measured throughout the layers of the network, allowing the analysis of the information flow [11]. Initially, it is assumed that input features X are obtained from a mapping of the target values Y. In sequence, the output of the ith hidden layer, h_i, is viewed as a transformation of the input of the preceding layer. Finally, the predicted output $\hat{Y}$ is generated on the output layer, as shown in Figure 2. Hence, the information flow can be measured considering the MI between the hidden layers and the input, $I(X, h_i)$, and the target outputs, $I(Y, h_i)$.

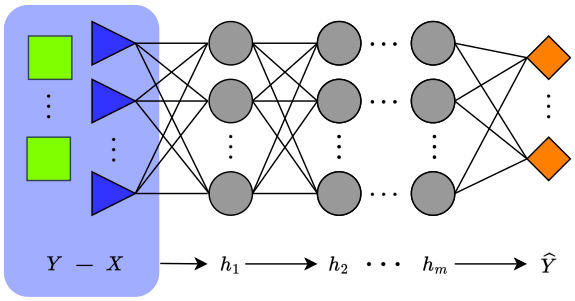

Fig. 2. An example of a neural network with m hidden layers, labels Y, an input layer X and an output layer $\hat{Y}$

Assuming the supervised paradigm, an immediate consequence is the Data Processing Inequality (DPI), which states that the information about Y lost in one layer cannot be recovered in subsequent layers [21]. Mathematically, for any $i \geq j$, it holds that

$$I(X, Y) \geq I(Y, h_j) \geq I(Y, h_i) \geq I(Y, \hat{Y}). \quad (9)$$

This implies that, by processing X, the information about the target value Y cannot be increased throughout the layers. However, in the context of RL, we often lack a reference or desired output value for each input, which implies that the application of DPI might not be feasible.

However both supervised and reinforcement learning paradigms can benefit from IB for the control of the information flow. Basically, the information flow from input to an intermediate point of the DNN can be constrained, e.g. through the application of a limit on $I(X, h_i)$ (in certain cases, it could be expressed in terms of the Kullback-Leibler divergence [21]). This constraint contributes for better generalization of the model in both paradigms.

On the other hand, considering the analysis of the information flow in DNNs, the Information Plane was proposed [12]. The main idea is to evaluate mutual information between input X and output Y through a specific layer h_i along the model training, allowing a deeper understanding of phases of learning. In the supervised case, the IP trajectory usually consists of two phases: empirical error minimization and representation compression [12]. , as illustrated in Figure 3. In the first phase, the empirical error is reduced, leading to an increase in the mutual information between the layers and the output $I(h_i, Y)$. Conversely, in the second phase, the input representation is compressed, decreasing the mutual

information between the input and the representation $I(X, h_i)$ while minimally affecting the mutual information between the layers and the output.

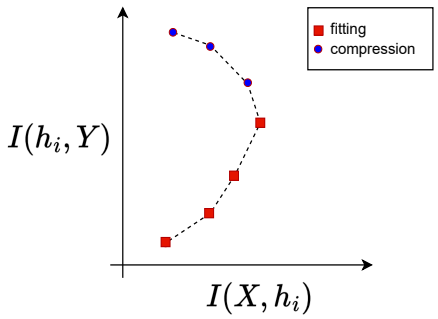

Fig. 3. Information Plane with mutual information dynamics during supervised learning.

It is important to highlight that, in supervised learning, mutual information is calculated based on the true output label Y [10]. This makes it difficult to replicate this analysis for reinforcement learning scenarios.

To address this, we propose a novel approach to analyze the information flow in RL. Our approach focuses on building a Information Plane by solely considering the variables from the input and from the neural network itself. Specifically, we investigate the relationships through MI between the input and the network's internal representations at different layers, $I(X, h_i)$, including the output of the network. However, the MI between the layers and the labels $I(Y, h_i)$ are disregarded, since Y is not available in RL. By examining how information is processed and transformed within the network, we can gain insights into the RL learning process and identify potential bottlenecks or inefficiencies.

IV. Simulation Scenario

The selected environment was CartPole, illustrated in the Figure 4. On this environment a pole is attached by a joint to a cart, which moves along a frictionless track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces to the left and right direction on the cart [22].

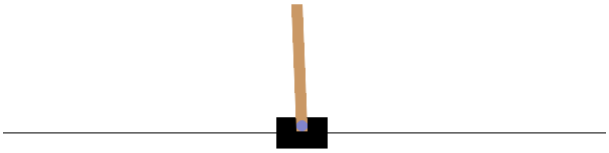

Fig. 4. Visualization of CartPole enviroment.

The action space $\mathcal{A}$ is limited to:

$$\text{Push cart to} \begin{cases} 0 : \text{left} \\ 1 : \text{right} \end{cases} \tag{10}$$

The observation space $\mathcal{S}$ has four elements that are arranged in Table I. The initial space $\mathcal{S}_0$ consists of 4 samples extracted from a uniform distribution $\mathcal{U}(-0.05, 0.05)$.

TABLE I
Environmental observations CartPole

Observation	Min	Max
Cart Position	-4.8	4.8
Cart speed	$-\infty$	∞
Pole angle	$\sim -24°$	$\sim 24°$
Pole angular velocity	$-\infty$	∞

The reward function R is $+1$ for each step. The criteria for ending the episode are:

- If the cart position exceeds the region $(-2.4, 2, 4)$;
- If the pole angle leaves the region $(-12°, 12°)$;
- If the accumulated reward reaches the limit of 500.

The code for the experiment was developed in Python. The main libraries used were Gymnasium [22], Stable-Baselines3 [23] and NPEET [24].

The Gymnasium library was used for its simulation environments.

The Stable-Baselines library was used by the structured reinforcement learning techniques, in which, as previously mentioned, the selected technique was Proximal Policy Optimization (PPO) [20].

The estimation of MI is made through the NPEET library, which is based on [25].

The hyperparameters of the implemented PPO agent are displayed in Table II. A complete description of the hyperparameters 'gamma', 'gae lambda', can be found in [4].

TABLE II
PPO Hyperparameters

Hiperparameter	Value
learning rate	3e-4
rollout size	2048
batch size	64
epochs	10
gamma	0.99
gae lambda	0.95
clip range	0.2
normalize advantage	True
Entropy coef. (c1)	0
Value funct. coef (c2)	0.5
max grad norm	0.5

The agent was originally trained for $1e5$ timesteps on a (0) seed resulting in 500 epochs. Subsequently, the training was extended to $1e6$ timesteps, resulting in 5000 epochs. The training took place in three stages: testing the agent, collecting observations and updating the networks.

In the testing stage, the agent is exposed to the environment in 100 seeds of the Fibonacci sequence. During each seed, the states, action distribution, and entropy of the action distribution

are collected. The action distribution is extracted directly from the latent space of the network. When the last seed is finished, the agent enters the collection stage. It is worth noting that the agent is not updated during this step.

In the collection step, the rollout is populated with state transitions (s_k, a_k, r_k, s_{k+1}). When reaching its maximum capacity, the agent enters the update phase. At this stage, a batch rollout scan is performed for each epoch. For each batch, the mutual information, control action, gradient and network weights are measured. At the end of all epochs, the rollout is cleaned and the process is repeated until the stipulated number of steps is reached.

For comparison purposes, the actions of the Linear Quadratic Regulator (LQR) control mechanism [26] from the implementation in [27] are also considered.

V. Analysis of IB and IP

Figure 5 illustrates the training loss components for the PPO agent in the CartPole environment. It is possible to observe a clear trend of decreasing loss across all components: entropy loss L_H, policy gradient loss $\mathcal{L}^{CLIP}$ and value loss L_ϕ. The fluctuations in the loss are common and highlight the importance of training with multiple seeds to ensure robust performance [28]. Around 200 epochs, the loss appears to converge, suggesting that the agent has learned a reasonably good policy. This statement is further supported by the trained agent achieving the maximum possible reward consistently during evaluation with multiple random seeds.

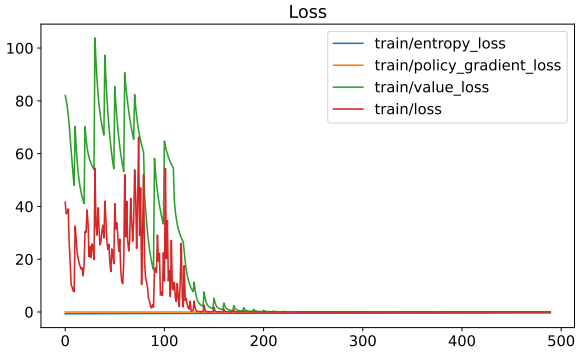

Fig. 5. Combined Loss with 500 epochs.

The two plots in Figure 6 show the evolution of gradients for the Policy network and Value network during training. This training was initially conducted for $1e5$ timesteps, and was later extended up to $1e6$ timesteps, but for graphical visibility purposes it was limited to the first 1000 epochs. In the phase before stabilization around 200 epochs, it is possible to observe a higher magnitude of gradients in the Value network compared to the Policy network. This suggests that the agent initially focuses on learning accurate state-value estimates (Critic) before refining its policy (Actor).

As training progresses, the gradients in the Policy network increase, indicating that the agent starts to actively adjust its policy based on the improved value estimates. While the Value network continues to exhibit relatively stable gradients, indicating consistent refinement of state value estimates, the Policy network demonstrates persistent gradient activity even after 500 epochs. This suggests that the agent continues to explore and refine its policy throughout the extended training period, adapting to the evolving dynamics of the environment and seeking to further improve its performance.

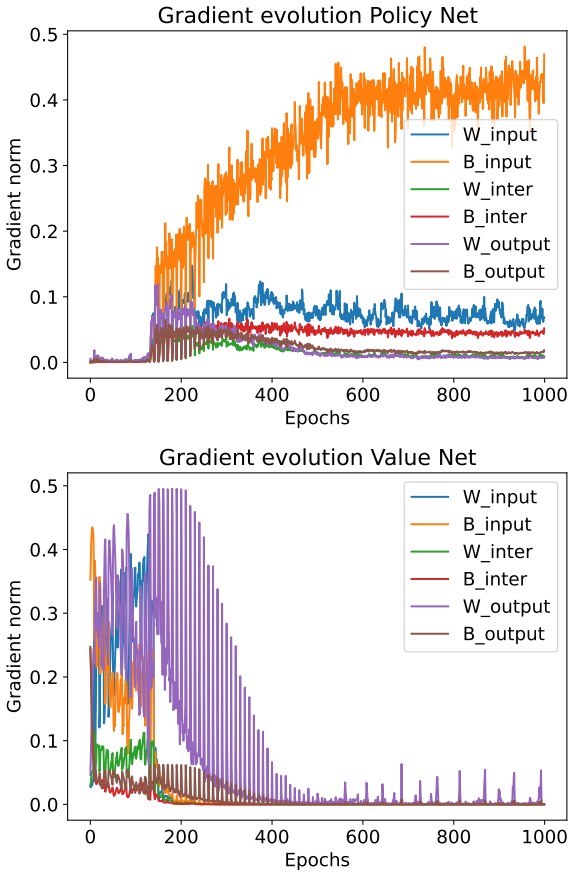

Fig. 6. Gradient evolution with 1000 epochs.

Figure 7 shows the evolution of the absolute values of the weights of both Policy and Value networks. It is evident that the absolute weights in both networks gradually increase during the initial phase of training, up to approximately 200 epochs. Beyond this point, the weights in the Value network exhibit a noticeable stabilization, suggesting that the network has converged to a stable set of parameters, with consistent estimated state values, given the optimal reward obtained. In contrast, the weights in the Policy network continue to display fluctuations even after 200 epochs. This persistent variability can likely be attributed to the inherent stochasticity of the environment, as well as the ongoing exploration-exploitation trade-off inherent in the policy optimization process. Such behavior underscores the dynamic nature of policy learning, where the network continuously adapts to refine its decision-making strategy.

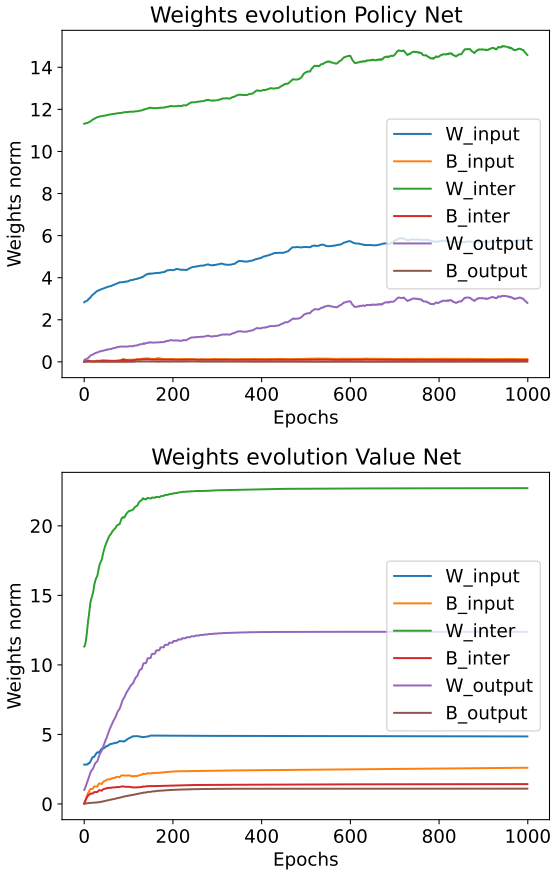

Fig. 7. Weights evolution with 1000 epochs.

To study the information flow evolution during the training, we analyze the Information Planes associated with the Policy network. First, we consider the IP $I(X, h_1) \times I(\hat{Y}, h_1)$, as shown in Figure 8, which compares the MI of the output of the first hidden layer to the input data X and to the output of the network $\hat{Y}$. An initial phase of empirical exploration is observed, with cluster of points in IP. Each cluster is associated with a similar policy. A significant change on the policy causes the creation of a new cluster. At this point, a gain on $I(\hat{Y}, h_1)$ is not consistent and, in the beginning of the training, $\hat{Y}$ tends to be bad choices. Also, different from the supervised case, no compression happens at this phase. Once training stabilizes around 200 epochs, a new phase begins and the first layer begins to expand the input data, leading to an increase in $I(X, h_1)$. Notably, this expansion occurs while maintaining a consistent relationship with $I(h_1, \hat{Y})$, suggesting that the network prioritizes feature transformation over compression during this stage of training.

In contrast, the PI $I(X, h_2) \times I(\hat{Y}, h_2)$ in Figure 9 reveals an inverse relationship between the two phases. Initially, in the first phase, a reduction in empirical error is observed ($I(\hat{Y}, h_2)$ increases), and with a significant update in the policy, it alternates to compression of the input data ($I(X, h_2)$ decreases). Shortly before the 200 epochs, when training

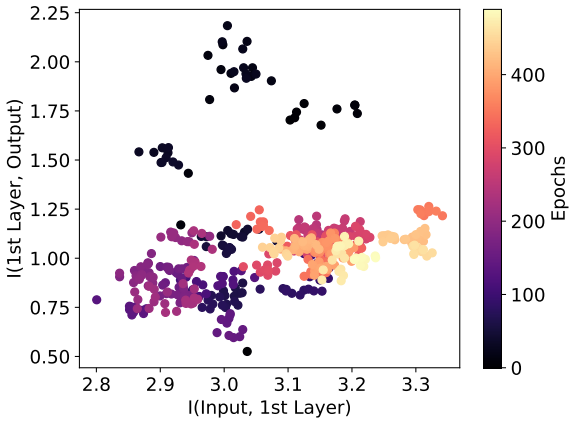

Fig. 8. IP $I(X, h_1) \times I(\hat{Y}, h_1)$ - Evolution of MI in first hidden layer.

stabilizes, $I(\hat{Y}, h_2)$ decreases, then increase significantly. This probably happens because $\hat{Y}$ starts to show intelligent behavior and this change on $\hat{Y}$ affects $I(\hat{Y}, h_2)$. After stabilization, the second phase occurs and the empirical error continues to decrease, leading to the significant increase on $I(\hat{Y}, h_2)$. This suggests that the second hidden layer alternates between error minimization and compression in the first phase, and then, on the subsequent phase, it focus on refining its predictive capabilities.

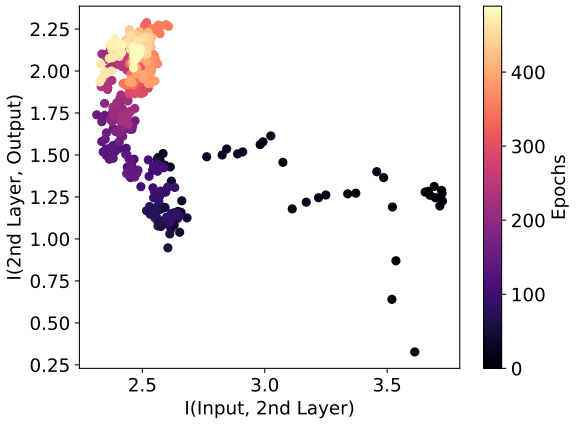

Fig. 9. IP $I(X, h_2) \times I(\hat{Y}, h_2)$ - Evolution of MI in second hidden layer.

We also consider the IP $I(X, h_1) \times I(h_1, h_2)$ in Figure 10, i.e. the evolution of mutual information between the initial layers. It reveals that, in the first phase (before 200 epochs), the information compression is dominant, with both $I(X, h_1)$ and $I(h_1, h_2)$ decreasing, persisting until the training stabilizes. After stabilization, the second phase takes place and the information expansion occurs, with both $I(X, h_1)$ and $I(h_1, h_2)$ increasing, but at different rates. This pattern highlights the initial layers transition from compression-driven learning to a phase of refinement and error reduction.

From another perspective, to emulate the original approach

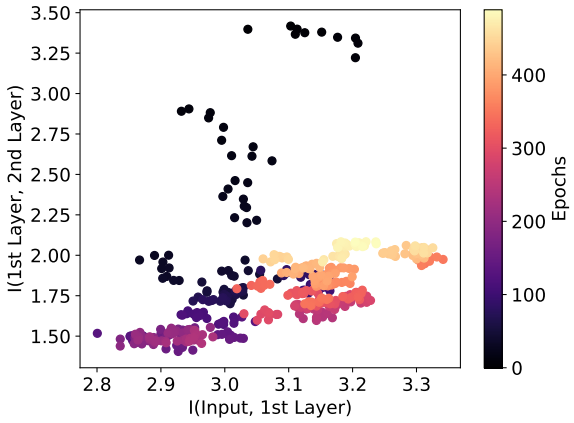

Fig. 10. IP $I(X, h_1) \times I(h_1, h_2)$ - Evolution of MI in first hidden layer..

for studying mutual information evolution in supervised learning, measurements were taken by comparing the Policy network output to an LQR control mechanism [26]. In other words, instead of the network output $\hat{Y}$, we compare the hidden layers output to the LQR control decision Y_C. However, we emphasize that this approach is an approximation, since the solution found by the agent after convergence is not the same as that of the LQR control.

Figure 11 illustrates the IP $I(X, h_1) \times I(Y, h_1)$. Initially, in the first phase, the clusters formation can still be observed. Again, no compression is noted, and a random behavior persists, until the training stabilizes around 200 epochs. Subsequently, in the second phase, the increase in $I(X, h_1)$ is present, but, different from $I(\hat{Y}, h_1)$, $I(Y, h_1)$ is clearly increasing. This suggests that the first hidden layer focus on error minimization.

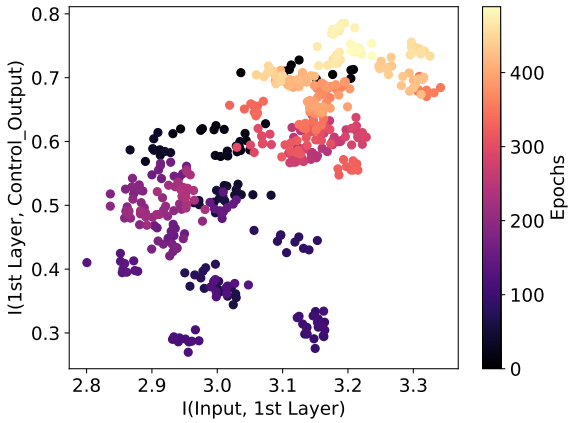

Fig. 11. IP $I(X, h_1) \times I(Y_C, h_1)$ - Evolution of MI in first hidden layer.

From IP $I(X, h_2) \times I(Y_C, h_2)$ in Figure 12, it is possible to observe that the first phase is basically characterized by data compression and a concurrent decrease in empirical error is observed ($\hat{Y}$ changes before a significant improvement), lasting until the training stabilizes around 200 epochs. In the subsequent phase, the focus on further minimization of the empirical error while maintaining the level of data compression achieved earlier is observed as well. This two-phase behavior underscores the network's ability to balance compression and error reduction during the learning process.

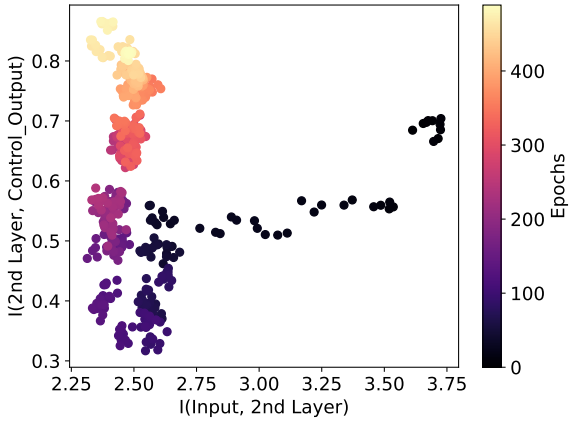

Fig. 12. IP $I(X, h_2) \times I(Y_C, h_2)$ - Evolution of MI in second hidden layer.

Comparing both cases, i.e. using $I(\hat{Y}, h_i)$ or $I(Y_C, h_i)$, the IP have similarities, which corroborates the use of IP based only on the input and outputs of the actor network in RL. Figures 8 and 11, which focus on the first hidden layer, agree in first phase, with no coherent compression; while, for the second phase, error minimization becomes more evident with $I(Y_C, h_1)$. However, a relative increase in $I(\hat{Y}, h_1)$ also noted. Comparing Figures 9 and 12, the information flow agrees on the second phase, but, in the first phase, $I(\hat{Y}, h_2)$ indicates minimization error and compression, while $I(Y_C, h_2)$ focus only on compression.

It is also worth mentioning that, differently from the supervised case, the hidden layers tends to first focus on compression and then on error minimization (fitting). In addition, the identification of two phases in RL also contributes to a set of novel possibilities to aid the training, in which we mention, for instance, the analysis of the overall progress of the training and definition of number of epochs.

VI. Conclusion

This work investigated the application of the IB principle and the Information Plane to analyze the learning dynamics of DRL agents, specifically using PPO in the CartPole environment. Our analysis revealed complex, layer-specific information flow patterns in the IP within the Policy network. We observed two distinct phases, in which information compression and error minimization within different layers of the network also occurs during the learning process, mirroring certain observations from supervised learning. However, the order of their occurrences are completely different. While some layers initially compressed information before expanding it as performance improved, others exhibited the opposite

trend. The hidden layer closer to the output of the network, for instance, first focuses on compression in the first phase, and the, on the second phase, on error minimization (the opposite order from supervised learning).

These findings suggest that different layers adapt uniquely to the task, highlighting the potential of the Information Plane as a tool for understanding and potentially optimizing DRL agent behavior. Future work could explore these dynamics in more complex environments and networks, and investigate methods to guide the learning process based on Information Plane trajectories.

ACKNOWLEDGMENT

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, ser. Adaptive computation and machine learning. Cambridge, Mass: MIT Press, 1998.

[2] M. Lapan, *Deep reinforcement learning hands-on: apply modern RL methods to practical problems of chatbots, robotics, discrete optimization, web automation, and more*, 2nd ed., ser. Expert insight. Birmingham Mumbai: Packt, 2020.

[3] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, Singapore: IEEE, May 2017, pp. 3389–3396. [Online]. Available: http://ieeexplore.ieee.org/document/7989385/

[4] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," Oct. 2018, arXiv:1506.02438 [cs]. [Online]. Available: http://arxiv.org/abs/1506.02438

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, ser. Adaptive computation and machine learning. Cambridge, Massachusetts London, England: The MIT Press, 2016.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: https://www.nature.com/articles/nature14236

[7] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van Den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017. [Online]. Available: https://www.nature.com/articles/nature24270

[8] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis, and D. Silver, "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II," 2019. [Online]. Available: https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/

[9] A. Kumar, V. Zhuang, R. Agarwal, Y. Su, J. D. Co-Reyes, A. Singh, K. Baumli, S. Iqbal, C. Bishop, R. Roelofs, L. M. Zhang, K. McKinney, D. Shrivastava, C. Paduraru, G. Tucker, D. Precup, F. Behbahani, and A. Faust, "Training Language Models to Self-Correct via Reinforcement Learning," Oct. 2024, arXiv:2409.12917 [cs]. [Online]. Available: http://arxiv.org/abs/2409.12917

[10] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," Apr. 2000, arXiv:physics/0004057. [Online]. Available: http://arxiv.org/abs/physics/0004057

[11] N. Tishby and N. Zaslavsky, "Deep Learning and the Information Bottleneck Principle," Mar. 2015, arXiv:1503.02406 [cs]. [Online]. Available: http://arxiv.org/abs/1503.02406

[12] R. Shwartz-Ziv and N. Tishby, "Opening the Black Box of Deep Neural Networks via Information," Apr. 2017, arXiv:1703.00810 [cs]. [Online]. Available: http://arxiv.org/abs/1703.00810

[13] P. Yingjun and H. Xinwen, "Learning Representations in Reinforcement Learning:An Information Bottleneck Approach," Nov. 2019, arXiv:1911.05695 [cs]. [Online]. Available: http://arxiv.org/abs/1911.05695

[14] X. Lu, K. Lee, P. Abbeel, and S. Tiomkin, "Dynamics Generalization via Information Bottleneck in Deep Reinforcement Learning," Aug. 2020, arXiv:2008.00614 [cs]. [Online]. Available: http://arxiv.org/abs/2008.00614

[15] J. Fan and W. Li, "DRIBO: Robust Deep Reinforcement Learning via Multi-View Information Bottleneck," in *Proceedings of the 39th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, Jul. 2022, pp. 6074–6102. [Online]. Available: https://proceedings.mlr.press/v162/fan22b.html

[16] S. J. Russell, P. Norvig, and E. Davis, *Artificial intelligence: a modern approach*, 3rd ed., ser. Prentice Hall series in artificial intelligence. Upper Saddle River: Prentice Hall, 2010.

[17] S. Ivanov and A. D'yakonov, "Modern Deep Reinforcement Learning Algorithms," Jul. 2019, arXiv:1906.10025 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1906.10025

[18] D. Dutta and S. R. Upreti, "A survey and comparative evaluation of actor-critic methods in process control," *The Canadian Journal of Chemical Engineering*, vol. 100, no. 9, pp. 2028–2056, Sep. 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/cjce.24508

[19] S. M. Kakade, "A Natural Policy Gradient," in *Advances in Neural Information Processing Systems*, T. Dietterich, S. Becker, and Z. Ghahramani, Eds., vol. 14. MIT Press, 2001.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," Aug. 2017, arXiv:1707.06347 [cs]. [Online]. Available: http://arxiv.org/abs/1707.06347

[21] T. M. Cover and J. A. Thomas, *Elements of information theory*, 2nd ed. Hoboken, N.J: Wiley-Interscience, 2006, oCLC: ocm59879802.

[22] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," Mar. 2023. [Online]. Available: https://zenodo.org/record/8127025

[23] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[24] gregversteeg, "GitHub - gregversteeg/NPEET: Non-parametric Entropy Estimation Toolbox," 2014. [Online]. Available: https://github.com/gregversteeg/NPEET

[25] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Physical Review E*, vol. 69, no. 6, Jun. 2004, publisher: American Physical Society (APS). [Online]. Available: http://dx.doi.org/10.1103/PhysRevE.69.066138

[26] A. Shaiju and I. R. Petersen, "Formulas for Discrete Time LQR, LQG, LEQG and Minimax LQG Optimal Control Problems," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 8773–8778, 2008. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1474667016403629

[27] JoKoum, "GitHub - JoKoum/reinforcement-learning-vs-control-theory: Agent versus Controller approach in balancing CartPole system." 2021, publication Title: GitHub. [Online]. Available: https://github.com/JoKoum/reinforcement-learning-vs-control-theory

[28] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning that Matters," Jan. 2019, arXiv:1709.06560 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1709.06560