Reinforcement Learning with Information-Theoretic Actuation

Elliot Catt[1*], **Marcus Hutter**[1,2], **Joel Veness** [2]

[1] Australian National University
[2] Deepmind
elliot.carpentercatt@anu.edu.com, www.hutter1.net, aixi@deepmind.com

Abstract

Reinforcement Learning formalises an embodied agent's interaction with the environment through observations, rewards and actions. But where do the actions come from? Actions are often considered to represent something external, such as the movement of a limb, a chess piece, or more generally, the output of an actuator. In this work we explore and formalize a contrasting view, namely that actions are best thought of as the output of a sequence of internal choices with respect to an action model. This view is particularly well-suited for leveraging the recent advances in large sequence models as prior knowledge for multi-task reinforcement learning problems. Our main contribution in this work is to show how to augment the standard MDP formalism with a sequential notion of internal action using information-theoretic techniques, and that this leads to self-consistent definitions of both internal and external action value functions.

Keywords

Reinforcement Learning, large action spaces, compression, coding, internal actions, sampling.

1 Introduction

It is hard to speak of embodied agents these days without mentioning or appealing to some notion of Reinforcement Learning. This particular mathematical formalism has been so successful of late that the validity of its various modelling assumptions rarely gets called into question. Yet recently we have seen a step-change in the capabilities of generative modelling, with the most striking example being in multi-modal language applications; the acquisition of gigantic multi-task datasets via internet scraping and scalable approaches to training has led to a renewed excitement for building next generation question-answering systems, chat bots, productivity tools, sentiment analysis, and in some circles, has even produced a newfound sense of optimism that the original goals of Artificial Intelligence may well be obtainable within our lifetimes.

Yet what does this mean for Reinforcement Learning? While its success in restricted domains is no longer in doubt, questions remain about its long-term viability as a foundational paradigm for Artificial Intelligence. For example, effective exploration, even in restricted settings such as finite MDPs, is problematic in large unstructured state spaces, with various lower bounds demonstrating polynomial dependence on the size of the state space, e.g. (Strehl, Li, and Littman 2009). While there are some noteworthy recent examples of hard exploration problems being overcome by clever heuristics (Ecoffet et al. 2021), the situation in general looks challenging, if not dire. On the other hand, recent advances in sequence modelling combined with the acquisition of gigantic datasets via internet scraping has led to a seeming step-change (Brown et al. 2020) in the ability of various types of probabilistic models to generate plausible continuations. Is there a way to leverage this, while keeping the basic reinforcement learning formalism and derived notions such as value functions, policies, return, etc intact?

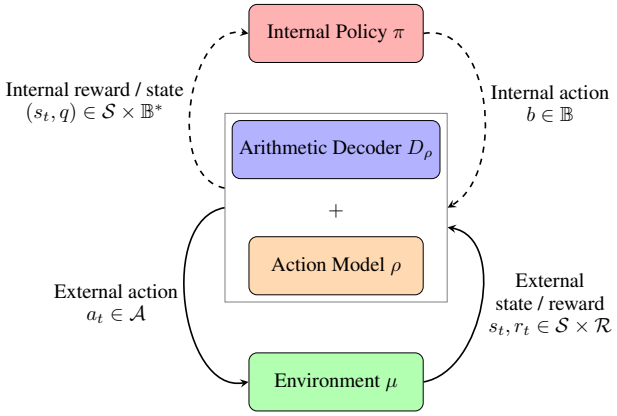

Figure 1: Agent-environment loop with internal actions.

Our proposal argues for rethinking the fundamental notion of action in reinforcement learning. Actions are often considered to represent something external, such as the movement of a limb, a chess piece, or more generally, the output of an actuator. In this work however, we develop a generic notion of *internal* action, which is implied by a choice of action model ρ. The key technical insight we leverage is the well-known duality between optimal lossless coding strategies and probabilities from information theory. At a high level, instead of an agent directly picking an action from the action space $\mathcal{A}$, instead it will pick a sequence of internal actions from an internal action set $\mathbb{B}$ which will de-

[*]Contact Author

code to an external action from $\mathcal{A}$. Figure 1 depicts this interaction graphically.

So what do we gain by introducing this particular layer of indirection in the agent's choice of action? Breaking up an action into a series of internal actions seems like a reasonable approach to dealing with large action spaces, and indeed has been used in other planning settings, but it immediately throws up a number of questions. How do we decompose an arbitrary action space? Is there a universal, or in some sense optimal decomposition? When should the agent stop generating internal actions and communicate an external action to the environment? Does this even make sense in a reinforcement learning setting? How do we leverage prior knowledge in the form of a default policy? Are there ramifications for multi-task RL? Can we efficiently compute or sample good actions? This paper will argue that our particular information-theoretic decomposition using an arithmetic decoder coupled with a coding distribution implied by a choice of action model naturally addresses all these questions, and opens up the possibly of leveraging recent advances in meta-learning and large-scale language/sequence models to deal with large problems using existing RL techniques.

Content. The paper is structured as follows: Section 2 reviews some background material and establishes some notation; Section 3 introduces internal actions, with Section 4 formally establishing the connection between internal/external agents and environments; Section 5 shows how the internal action framework naturally accommodates multi-task reinforcement learning settings with different action spaces. We conclude with an extended discussion in Section 6 and cover related and future work in Sections 7 and 8.

2 Preliminaries

We now briefly review the necessary background material required to describe our internal action agent-environment interaction loop.

Sequential Prediction. A finite alphabet $\mathcal{X}$ is a set of symbols. A string of symbols $x_1 x_2 \dots x_n \in \mathcal{X}^n$ of length n is denoted by $x_{1:n}$. The prefix $x_{1:j}$ of $x_{1:n}$, $j \leq n$, is denoted by $x_{\leq j}$ or $x_{<j+1}$. The empty string is denoted by ϵ. The set of strings whose symbols come from the alphabet $\mathcal{X}$ with length at most n is defined by $\mathcal{X}^{\leq n} := \{\epsilon\} \cup \bigcup_{i=1}^{n} \mathcal{X}^i$. The set of strings of symbols from alphabet $\mathcal{X}$ with finite length is denoted by $\mathcal{X}^* := \{\epsilon\} \cup \bigcup_{i=1}^{\infty} \mathcal{X}^i$. The concatenation of two strings x and y is denoted by xy. The length of a string x will be denoted by $|x|$. We will use $y \in x$ to denote that the symbol y is in the string x.

A (coding) distribution ρ is a sequence of probability mass functions $\rho_n : \mathcal{X}^n \to [0,1]$, which for all $n \in \mathbb{N}$ satisfy the constraint that $\rho_n(x_{1:n}) = \sum_{y \in \mathcal{X}} \rho_{n+1}(x_{1:n} y)$ for all $x_{1:n} \in \mathcal{X}^n$, with the base case $\rho_0(\epsilon) := 1$. From here onwards, whenever the meaning is clear from the argument to ρ, the subscript on ρ will be dropped. Under this definition, the conditional probability of a symbol x_n given previous data $x_{<n}$ is defined as $\rho(x_n|x_{<n}) := \rho(x_{1:n})/\rho(x_{<n})$ provided $\rho(x_{<n}) > 0$, with the familiar chain rules $\rho(x_{1:n}) = \prod_{i=1}^{n} \rho(x_i|x_{<i})$ and $\rho(x_{j:k} \,|\, x_{<j}) = \prod_{i=j}^{k} \rho(x_i|x_{<i})$ now following. We will use $\Delta(\mathcal{X})$ to denote the space of probability distributions over $\mathcal{X}$.

Arithmetic Encoding / Decoding. A fundamental technique known as *arithmetic encoding* (Rissanen and Langdon 1979; Witten, Neal, and Cleary 1987) makes explicit the connection between coding distributions and source codes. Binary arithmetic encoding is a general purpose parameterized technique that takes in a distribution ρ (known as a coding distribution) and some data $x_{1:n} \in \mathcal{X}^n$, and produces a uniquely decodable binary codeword $C_\rho(x_{1:n}) \in \{0,1\}^*$, whose length is essentially $\lceil -\log_2 \rho(x_{1:n}) \rceil$, which is optimal in terms of expected length if the data is sampled from ρ. In essence, shorter binary codewords are assigned to data which has a higher chance of occurring under ρ, and longer binary codewords are assigned to the less probable data items. Arithmetic decoding is the reverse of this procedure; it takes a coding distribution ρ, a binary code word $y_{1:k} = C_\rho(x_{1:n})$, and returns the original data $D_\rho(y_{1:k}) = x_{1:n}$. We will also use the shorthand notation $D_\rho(y_{1:k} \,|\, s) := D_{\rho(\cdot|s)}(y_{1:k})$ to denote decoding with respect to a coding distribution conditioned on the string s. We refer the reader to the standard text of (Cover 1999) for further information.

Markov Decision Processes. A Markov Decision Process (MDP) is a type of probabilistic model widely used within reinforcement learning (Sutton and Barto 2018; Szepesvári 2010) and control (Bertsekas and Tsitsiklis 1996). In this work, we limit our attention to finite horizon, time-homogeneous MDPs whose action and state spaces are finite. Formally, an MDP is a quadruplet $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mu)$, where $\mathcal{S}$ is a finite, non-empty set of states, $\mathcal{A}$ is a finite, non-empty set of actions, $\mathcal{R} \subset \mathbb{R}$ is the reward space, and μ is the transition probability kernel that assigns to each state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$ a probability measure $\mu(\cdot \,|\, s, a)$ over $\mathcal{S} \times \mathcal{R}$. $\mathcal{S}$ and $\mathcal{A}$ are known as the *state space* and *action space* respectively. The transition probability kernel gives rise to the *state transition kernel* $\mathcal{P}(s'|s,a) := \mu(\{s'\} \times \mathcal{R} \,|\, s, a)$, which gives the probability of transitioning from state s to state s' if action a is taken in s.

An agent's behavior is determined by a *policy* that defines, for each state $s \in \mathcal{S}$ and time $t \in \mathbb{N}$, a probability measure over $\mathcal{A}$ denoted by $\pi_t(\cdot \,|\, s)$. A *stationary policy* is a policy which is independent of time, which we will denote by $\pi(\cdot \,|\, s)$ where appropriate. At each time t, the agent communicates an action $A_t \sim \pi_t(\cdot \,|\, S_{t-1})$ to the system in state $S_{t-1} \in \mathcal{S}$. The system then responds with a state-reward pair $(S_t, R_t) \sim \mu(\cdot \,|\, S_{t-1}, A_t)$. Here we will assume that each reward is bounded between $[r_{\min}, r_{\max}] \subset \mathbb{R}$ and that the system starts in a state s_0 and executes for an infinite number of steps. Thus the execution of the system can be described by a sequence of random variables $S_0, A_1, S_1, R_1, A_2, S_2, R_2, ...$

The finite m-horizon *return* from time t is defined as $Z_t := \sum_{i=t}^{t+m-1} R_i$. The expected m-horizon return from time t, also known as the *value function*, is denoted by $V_\mu^\pi(s_t) := \mathbb{E}[Z_{t+1} \,|\, S_t = s_t]$. The return space $\mathcal{Z}$ is the set of all possible returns. The *action-value function* is de-

fined by $Q^{\pi}_{\mu}(s_t, a_{t+1}) := \mathbb{E}[Z_{t+1} \mid S_t = s_t, A_{t+1} = a_{t+1}]$. An *optimal policy*, denoted by π^*_{μ}, is a policy that maximizes the expected return $\mathbb{E}[Z_{t+1} \mid S_t]$ for all t.

3 Information-Theoretic Actuation – Internal Actions

We now describe in detail how to combine the aforementioned building blocks into the internal reinforcement learning framework described in Figure 1, and discuss its ramifications. Compared with the standard agent-environment loop, there are two additional components with this setup: a choice of action model ρ, and an associated arithmetic decoder D_ρ that uses ρ as a coding distribution. The internal action space $\mathbb{B}$ is defined by the associated decoding alphabet used by D_ρ; for example, using a binary arithmetic decoder would lead to an internal action space of $\mathbb{B} = \{0, 1\}$. For pedagogical purposes, we will restrict our attention to this case in the rest of the paper, but remark that any finite decoding alphabet can in principle be used with our construction.

We first introduce our notion of internal action. At a high-level, one should think of a single internal action as a bit-commitment towards a particular choice of external action, with particular sequences of these corresponding to external actions. In a sense, internal actions correspond to a period of private deliberation by the agent, which upon conclusion produces a string describing the desired actuation in compressed form; in essence, the arithmetic decoder functions as a universal actuator, whose behavior can be completely configured by a choice of action model.

Reshaping of the action space. As alluded to before, the effect of the action model is to reshape the action space, which the following example will make clear. Figure 2 shows an illustrative example of the behavior of a binary arithmetic decoder equipped with an action model based on a GLN-based context mixing language model (Veness et al. 2019) that has been pre-trained on 9MB of grandmaster games in PGN (Portable Game Notation) format. On the left hand side of the table, we have the input to the decoder, and on the righthand side we have the decoded output; if we consider the first row, the LHS corresponds to the bitstring $10 = C_\rho(\text{a6})$ and the RHS corresponds to $D_\rho(10)$, with ρ here denoting our pre-trained language model. The LHS of the first 4 rows shows the encoding of a natural sequence of continuing moves (known as the Morphy Defense), while the last four rows show an illogical continuation of moves which ignore development, lose castling rights, and even hang the queen. One can see that much shorter codes are assigned to the more logical sequence of moves. This shows the effect of the action model as providing a type of inductive bias, which we will discuss in greater depth later.

In contrast, one could also consider the effect of a completely uninformative action model, $\rho_{\text{UNIFORM}}(a|s) := 1 \,/\, |\mathcal{A}|$, which assigns uniform probability mass to each possible external action in every state. Here every single action would have the same codelength of $\lceil \log |\mathcal{A}| \rceil$, which would correspond to a naive binarization of the external action space.

Input bits	Decoded Output
10	a6
10010	a6 Ba4
100100	a6 Ba4 Nf6
1001010111	a6 Ba4 Nf6 O-O
010010101010011	Nh6
01001010101010000110000010	Nh6 Kf1
0100101010101000110000011001001010	Nh6 Kf1 Qg5
010010101010100011000001100100101010100100010001	Nh6 Kf1 Qg5 Na3

Figure 2: Arithmetic decoding example. Some example decoded outputs from a pre-trained model on chess, with the model's context set to the Ruy Lopez opening, namely: *e4 e5 Nf3 Nc6 Bb5*.

When to stop decoding. Figure 2 also highlights a technical issue which we need to resolve, namely, how and when is a decoded action to be transmitted to the external environment? For example, if we wanted a chess-playing agent whose action space was the space of single moves, we need some way to know when our decoded output should be communicated to the environment as an external action. Although other solutions are possible, in this work we adopt the convention that every external action can be described as a string formed by the concatenation of atomic symbols from a common alphabet. More formally, we assume that the action space $\mathcal{A} \subseteq \mathbb{A}^{\leq k}$, where $\mathbb{A}$ denotes the sub-action alphabet, and k is a positive constant. We assume that the sub-action alphabet always contains a privileged termination symbol $\top \in \mathbb{A}$, which has the semantics that when it is decoded it causes an external action to be communicated to the environment. Note that in finite action/state MDPs, this modification does not impose any restrictions nor add further expressive power. Returning to the example shown in Figure 2, by identifying the space character with $\top$, we would know when to transmit an external action. This is implemented formally via a function $\tau : \mathbb{A}^{\leq k} \to \mathcal{A}$ which takes actions and returns the action component up to but not including the first $\top$, for example $\tau(\text{a6}\,\top) = \text{a6}$. This importantly handles the case of multiple $\top$ symbols, for example $\tau(\text{a6}\,\top\,\text{Ba4}\,\top) = \text{a6}$.

A terminal symbol is not the only way to know when to stop decoding. Another approach could be to only allow prefix-free codes. This will however run into it's own problems, such as what prefix-free encoding to use, how to enumerate the elements of $\mathcal{A}$ so that the corresponding prefix code can be found easily (and vice versa). Using an "optimal" prefix code would require the use of universal Turing machines and is beyond the scope of this paper. Another choice to stop decoding is to consider the action before the last $\top$ symbol, instead of before the first. In this case the agent may take multiple actions without knowing the state in between them.

Internal action loop. External action selection is determined by executing our internal policy π until the concatenation of these binary actions uniquely decodes into an external action. Once the action model and arithmetic decoder have generated an external action, this external action will be sent to the external environment. The external environment will then return an external observation/reward to the action model and arithmetic decoder combination, and the

internal policy receives a reward r_t from the external environment. This interaction is displayed graphically in Figure 1 and described procedurally by Algorithm 1.

Algorithm 1: Internal Agent-Environment loop

Require: Internal policy $\pi : \mathcal{S} \times \mathbb{B}^* \to \Delta\mathbb{B}$
Require: External environment $\mu : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S} \times \mathcal{R})$
Require: Action model $\rho : \mathcal{S} \to \Delta\mathbb{A}^{\leq k}$

for $t = 1, 2, 3, \ldots$ **do**
  Observe $s_t, r_t \sim \mu(\cdot, \cdot | s_{t-1}, a_{t-1})$
  $a_t, q = \epsilon$
  while $\top \notin a_t$ **do**
    $b \sim \pi(\cdot | s_t, q)$
    $q = qb$
    $a_t = D_\rho(q | s_t)$
    $r_t = 0$
  end while
  $a_t = \tau(a_t)$
  Act a_t
end for

Example. We conclude this section with an example execution in the context of our previous chess example. Here the state space is the finite history of moves. At the first time-step, the system receives an observation from the external environment. This observation becomes the current state $s_0 =$ e4 e5 Nf3 Nc6 Bb5. Then given this state the internal policy takes internal binary actions one by one $b_0 = 1, b_1 = 0$. The internal policy continues to take internal binary actions until the sequence of internal binary actions uniquely decodes into an external action a_t. A concatenation of characters is an external action when it has a "terminal" symbol $\top$, a6 $= D_\rho(b_0 b_1 | s_0)$. Now that we know the external action, it is sent to the external environment μ, $a_1 =$ a6 . Then we receive the next observation from μ,

$$o_1 \sim \mu(\cdot | s_0 a_1) = \mu(\cdot | \text{e4 e5 Nf3 Nc6 Bb5 a6})$$

and now $s_1 := s_0 a_1 o_1$ and the process repeats.

4 Connecting External to Internal Agents and Environments

In this section, we will describe formally how to augment an arbitrary external environment to an internal environment that the internal action agent is able to interact with; additionally if the external environment is Markovian then the internal environment will also be Markovian. Our approach will be to construct an augmented environment ϑ, called the internal environment, comprised of the true environment, the action model and the arithmetic decoder. We will also show how the internal action agent can be uplifted to an external agent, and then show that both the internal environment with an internal action agent and the external environment with the uplifted policy are equivalent in the sense that they achieve identical action-value functions. These results will allow for easier analysis of the internal action agent setup, as well as the ability to apply any result or algorithm specific to MDPs to the internal agent setup.

Internal Environment. The internal environment ϑ is a stochastic function over internal states and internal actions to internal states and rewards. The internal state space used here will be $\mathcal{I} := \mathcal{S} \times \mathbb{B}^{\leq n}$, the state from the external environment and previous internal actions taken by the internal agent, until they are decoded to an external action. We consider the finite set $\mathbb{B}^{\leq n}$ over the infinite set $\mathbb{B}^*$, as for any external action a with $\rho(a) > 0$ there will always be a finite number of binary actions needed to decode a; n is the maximum of those finite numbers. We will use $\top$ to denote the "terminal" symbol, that is, the symbol that indicates when the concatenation of internal actions corresponds to a complete external action, and is sent to the external environment. We will use the symbols s, s' for elements of $\mathcal{S}$, the first component of the internal state. We will use q, q' for elements of $\mathbb{B}^{\leq n}$, the second component of the internal state, the internal agent's previous internal actions. The symbol b will be used for the internal agent's internal action. The symbol a will be used for a decoded external action, e.g. $D_\rho(qb|s) = a$. The true external environment will be denoted by μ, which is a stochastic function from external states and external actions to external states and rewards. The external state space is $\mathcal{S}$. The external action space is $\mathcal{A} \subseteq \mathbb{A}^{\leq k}$.

Definition 1 (MDP (Internal) Environment ϑ). *The internal policy π interacts with an internal environment $\vartheta : \mathcal{I} \times \mathbb{B} \to \Delta(\mathcal{I} \times \mathcal{R})$ which is defined by the action model ρ (encoder/decoder C_ρ/D_ρ generated by ρ) and the true external environment μ as follows:*

$$\vartheta(s'q'r | sq, b) := \begin{cases} \mu(s'r | s, \tau(a)) & \text{if } q' = \epsilon \ \wedge (\top \in a), \\ 1 & \text{if } s' = s \ \wedge q' = qb \wedge r = 0 \ \wedge (\top \notin a), \\ 0 & \text{otherwise} \end{cases}$$

where $a := D_\rho(qb|s)$, $(s'q', r) \in \mathcal{I} \times \mathcal{R}$, $sq \in \mathcal{I}$ and $b \in \mathbb{B}$.

The definition of ϑ is split up into three cases: In the first case the decoded qb contains the symbol $\top$, $\top \in a$ where $a := D_\rho(qb|s)$, and the previous binary characters q' resets to being the empty string ϵ. In this case the τ of the decoded action $D_\rho(qb|s)$ is sent to the external environment μ, and the next state s', is the external state s'. The second case of ϑ is when the internal agent is still decoding, that is, $\top \notin a$ and the next state $s'q' = sqb$ is updated by the agent's action b, and the internal reward r is 0. In the third case, where neither set of above conditions is satisfied, the probability of the state $s'q'$ and reward r is 0. In this way the environment ϑ is deterministic during the decoding process, and only stochastic when it sends the decoded action to the external environment.

Given the internal agent's policy π and the arithmetic decoder D_ρ, we can construct an external policy Π which will interact with the true external environment μ. The external policy Π is a stochastic function from external states $s \in \mathcal{S}$ to external actions $a \in \mathcal{A}$. To construct Π, we consider all possible binary strings $q \in \mathbb{B}^{\leq n}$ such that the arithmetic decoder will decode q into a given s. For this we will need to

define a *decodable* subset of $\mathbb{B}^{\leq n}$. We will use $\mathbb{D}$ to denote the set of decodable binary strings. A string q is decodable if $\top$ is in the decoding of the string, and $\top$ is not in the decoding of the first $|q|-1$ elements of the string. Formally this means

$$\mathbb{D}_s := \left\{ q \in \mathbb{B}^{\leq n} : \top \in D_\rho(q|s) \ \wedge \ \top \notin D_\rho(q_{<|q|}|s) \right\}.$$

We then consider the probability that π will output the internal binary actions that eventually construct q, which using the chain rule we can write as the product of probabilities that π will take the action of each element of q given the previous elements of q. All together this is written as follows:

$$\Pi(a|s) := \sum_{\substack{q \in \mathbb{D}_s: \\ a = \tau(D_\rho(q|s))}} \prod_{i=1}^{|q|} \pi(q_i | s q_{<i}). \tag{1}$$

It is important to note that there may be more than one binary string $q \in \mathbb{D}_s$ such that $a = \tau(D_\rho(q|s))$; this comes from how arithmetic decoders work. For example, consider a case where

$$\begin{aligned} &D_\rho(10|s) = e,\ D_\rho(100|s) = e4,\ D_\rho(101|s) = e4 \\ &D_\rho(1000|s) = e4\,c5,\ D_\rho(1001|s) = e4\,\top \\ &D_\rho(1010|s) = e4\,\top,\ D_\rho(1011|s) = e4\,e5 \end{aligned}$$

We have that both 1001 and 1010 are elements of $\mathbb{D}_s$ and both $\tau(D_\rho(1001|s)) = e4$ and $\tau(D_\rho(1010|s)) = e4$, therefore $\Pi(e4|s)$ would be a sum over 1001 and 1010.

Self-consistency of internal and external Q-values. We can use the external agent Π to interact with the external environment μ, just as any regular RL agent would.

Theorem 2 (Internal/External value equivalence)**.** *For all states $s \in \mathcal{S}$, previous internal actions $q \in \mathbb{B}^{\leq n}$, external actions $a \in \mathcal{A}$ and internal actions $b \in \mathbb{B}$, if $\tau(D_\rho(qb|s)) = a$ then*

$$Q^\Pi_\mu(s,a) = Q^\pi_\vartheta(sq, b). \tag{2}$$

That is, the action-value function for the external policy Π and external environment μ is equal to the action-value function for the internal policy π and the internal environment ϑ.

Proof. This proof comes from expanding the action-value function using Equation 1 and Definition 1 to rearrange the expanded action-value function. For the full proof see the supplementary material. □

Because of Equation 2 we are able to say that if an internal agent π performs well, in the sense of a high action-value, in the internal environment ϑ, then the uplifted version of the agent Π, performs well in the true external environment μ.

5 A Universal Action Interface for Multi-task RL

A key complication and limiting factor in the design of any multi-task RL system is how to deal with the potentially radically different action spaces required for each distinct task. While it is feasible to make a generic agent work well across multiple similar domains e.g. Atari games (Mnih et al. 2015), the situation becomes considerably more complicated when the action spaces of the different tasks vary dramatically. The arithmetic encoding-based approach we advocate provides an elegant solution to this problem, which builds on techniques from universal source coding.

Given $K > 1$ coding distributions, it is straightforward to combine them into a universal ensemble whose compression performance will be close to that of the best coding distribution in hindsight. If we denote the ith coding distribution by ρ_i, one can take a uniform Bayesian mixture of the K coding distributions, whose marginal distribution over sequences is given by

$$\xi(x_{1:n}) := \sum_{i=1}^{K} \frac{1}{K}\, \rho_i(x_{1:n}). \tag{3}$$

A standard dominance argument shows that the logarithmic loss/coding length of the mixture ξ compared to any choice of action model j is bounded by

$$\begin{aligned} -\log \xi(x_{1:n}) &\leq -\log\left(\frac{1}{K}\,\rho_j(x_{1:n})\right) \\ &= -\log \rho_j(x_{1:n}) + \log K, \end{aligned}$$

or in other words, the excess log-loss is bounded by a constant, which is asymptotically negligible when one considers the time-averaged performance of the ensemble.

This has important ramifications for multi-task reinforcement learning in our internal action formulation. Recently, various works (Janner, Li, and Levine 2021) have attempted to frame reinforcement learning in terms of probabilistic sequence models over interaction strings, i.e. defining a sequential probability measure ν over strings that represent state/reward/action histories in the form $s_1 r_1 a_1 \ldots$. By taking a uniform Bayesian mixture over multiple instances of these history-based measures for different tasks, just as in the coding distribution example, one also obtains a sequence model that is universal across all of these tasks. More formally, given a history string h which is an element of $(\mathcal{S}\times\mathcal{R}\times\mathcal{A})^* \cup ((\mathcal{S}\times\mathcal{R}\times\mathcal{A})^* \times (\mathcal{S}\times\mathcal{R}))$, we can define the uniform Bayesian mixture

$$\xi(h) = \sum_{i=1}^{K} \frac{1}{K}\, \nu_i(h) \tag{4}$$

over K history based measures ν_i, with each ν_i corresponding to a task specific history model. Note that this formulation in terms of measures on strings still implies the usual Bayesian learning in terms of sequential updating of the posterior, it is just hidden in this notation; see Section 2 by Milan et al. (2016) for a brief overview.

An interesting effect now emerges if we use the conditional action distribution $\xi(\cdot|s_1 r_1 a_1, \ldots, s_n r_n)$ as the action model in our setup. In particular, this action model will rapidly learn to *automatically generate actions appropriate for the underlying task*, without requiring any task identity information. How this works is subtle; Bayesian inference is used implicitly by ξ to determine which task the agent is most likely in, and due to the rapid convergence of the Bayesian mixture to the best task specific model, the action

model used for decoding after a small number of external environment interactions will essentially behave the same as if we knew which task specific action model to use in the first place. In other words, what this means in practice is that one can use ξ as the action model, and C_ξ will produce codes which are almost as short as any task-specific action encoding C_{ρ_j}. In particular, this implies that short bitstrings can decode to very different external actions which are plausible under either task-specific model.

The most interesting aspect about this construction is that the internal action formalism allows us to treat a multi-task reinforcement problem as a single reinforcement learning task with a common action space.

6 Discussion

This section discusses some interesting and potentially surprising ramifications of information-theoretic internal actions.

Uninformative internal policy π and application to Large Language Models. An interesting corollary of the internal reinforcement learning setup is that a uniform policy over internal actions gives rise to an external agent that selects actions that are essentially distributed to the action model. This is a by-product of the duality between optimal codes and probabilities, and can be seen with the following argument. In the case of a binary internal action space, a uniform policy will generate a particular sequence $b_{1:n}$ with probability 2^{-n}. Now, notice that the probability of an action $a \in \mathcal{A}$ in state s where $C_\rho(a|s) = b_{1:n}$ is given by

$$\rho(a|s) = 2^{\log_2 \rho(a|s)} \approx 2^{-|C_\rho(a|s)|} = 2^{-n}.$$

The approximate equality step is due to the small gap between the optimal code length $-\log_2 \rho(a|s)$ and the realised code length $C_\rho(a|s)$ produced by an arithmetic encoder coupled to ρ. The size of this gap is bounded by 1, but is essentially negligible for the purposes of this argument.

This has interesting ramifications for constructing agents when the action model is already useful, such as in the case of Large Language Models (LLMs) obtained by supervised learning on massive amounts of internet data. Random behaviour by the internal policy in this case will still produce useful behavior, which provides a natural starting point for any internal policy learning technique. In particular random internal-action exploration becomes targeted, possibly leading to optimal external-action exploration, in a similar fashion to Thompson Sampling. It is in this way that the action model provides a powerful mechanism for specifying data-dependent prior knowledge to existing reinforcement learning algorithms.

Specifying the action space from data. In complicated environments, it may be difficult or complicated to precisely specify the action space explicitly. This situation readily arises in natural language domains for example. In these cases it is more natural to simply learn a probabilistic model of the domain. Our internal agent formalism directly allows for this possibility via the action model. The action model allows for a strict separation between pre-training on data, for example pre-training an action model using a collection of grandmaster games in chess, and the resultant learning behavior of the internal agent.

It is also worth pointing out an interesting connection to meta-learning with sequence models across many tasks. Perhaps surprisingly, perplexity-based meta-learning of history-dependent LLMs is closely related to the explicit Bayesian mixture solution described in Equation 4. In particular, one can show that in many standard meta-learning setups, the optimal perplexity-minimizing solution is *exactly* a Bayesian mixture distribution (Ortega et al. 2019). Provided that a sufficiently powerful history-dependent model is used (such as the case with LLMs based on Transformers) to model the interaction histories, a low-perplexity solution can be seen as a learnt approximation to the explicit Bayesian construction we provided in in Equation 4. In this way the action space for a multi-task agent can be learnt directly from data alone, which goes some way to explaining the recent empirical success of approaches such as Janner, Li, and Levine (2021). In other words, if one wanted an agent that could play both chess and something with a radically different action space such as the text-based NetHack (Küttler et al. 2020), a natural strategy would be to pre-train using meta-learning a sequence model from example trajectories in both games, and use this to define the action model; the internal action framework will then automatically deal with the different underlying action spaces.

Comparison to binarization. It is instructive to consider the differences between a direct binarization of the action space compared with our approach. One can interpret the combination of an action model and a binary arithmetic decoder as a generalized form of binarization. As discussed earlier, an action model which assigned a uniform distribution over the action space in every state is equivalent to a naive binarization, with every action being assigned a code-length of $\lceil \log_2 |\mathcal{A}| \rceil$. Binarization of actions in reinforcement learning has the obvious benefit of reducing the size of the action space, which can lead to some benefits (Majeed and Hutter 2020). However, often this binarization comes with a corresponding increase to the planning horizon, and in many circumstances provides no benefits.

Our non-uniform binarization essentially reshapes the action space according to the knowledge contained within the action model. Thus planning using various types of depth limited search takes on a different meaning in our internal reinforcement learning setting. Although the planning algorithm may only be searching d steps ahead, the implied information-theoretic planning horizon might be much greater than d.

Computational advantages. Many reinforcement learning techniques require an ability to efficiently generate a random sample from the action space. A convenient property of our formalism is that it provides a generic technique to generate samples from arbitrary action models/action spaces. This is a byproduct of having an arithmetic decoder coupled to an action model. By generating a sequence of bits $y_{1:m}$ with each bit sampled from a Bernoulli$(1/2)$ distribution, and feeding them to a binary arithmetic decoder D_ρ

coupled to the action model ρ, one can show that external action $a := D_\rho(y_{1:m})$ is distributed according to ρ (MacKay 2003), which resembles Thompson sampling. We can also efficiently compute the probability of a as a product of conditional probabilities ($P[a] = \prod_{t=1}^{k} P[b_t|b_{<t}]$) required for some learning algorithms. We can even efficiently compute the cumulative probability based on the recursion $P(X_{1:k} \leq b_{1:k}) = [\![b_1 = 1]\!]P(X_1 = 0) + P(X_{2:k} \leq b_{2:k}|X_1 = b_1)P(X_1 = b_1)$. Unfortunately binarization does *not* lend itself to an efficient way of computing the most probable (MAP) action $\arg\max_a P(a)$. But Thompson sampling for large spaces often performs better than MAP anyway, since the latter is not representation invariant and favors brittle solutions. Binarization decreases the branching factor in planning algorithms but increases the planning horizon. Since binarized actions are length-optimized this *may* still lead to a net win. For example, depth-limited planning techniques, which typically have an exponential dependence on the length of the horizon, now have an exponential dependence on the combined code-length under ρ, which drastically alters their semantics and is closely connected to using a prior policy to guide search such as in successful approaches for Computer Go (Silver et al. 2017; Orseau et al. 2018; Orseau and Lelis 2021).

Pre-training and universality. A common use case in machine learning is to consider fine tuning an existing pre-trained model to save on compute. The next result shows that pre-training on any data will not affect the asymptotic performance of any consistent density estimator. In our context, it suggests that a good general approach to constructing an action model for a new domain might be to first pre-train on large, task-agnostic data and then to use fine tuning to incorporate task-specific knowledge if this data is available.

More formally, consider sequences $X_{1:\infty}$ over a finite alphabet $\mathcal{X}$ sampled from P_{θ_0}. Assume $\theta(X_{1:n})$ is a consistent estimator of θ_0. Then whatever the first k samples $x_{1:k}$ are, $\theta(x_{1:k}X_{k+1:n})$ is still a consistent estimator of θ_0. Additionally, the reverse is also true. Most importantly, this holds without *any* assumptions on the stochastic process $(X_t) \sim P_{\theta_0}$.

Proposition 3 (consistency is immortal). *For any fixed $k \in \mathbb{N}$, $\theta(X_{1:n})$ is a consistent estimator of θ_0 if and only if for all $x_{1:k}$ such that $P_{\theta_0}[x_{1:k}] > 0$, $\theta(x_{1:k}X_{k+1:n})$ is a consistent estimator of θ_0.*

One consequence of this proposition is that for a given ρ and π, if Π defined in Equation 1 is a consistent estimator of π^*_μ, the optimal policy in μ, then if the action model ρ is pre-trained on additional data then Π is still a consistent estimator.

Discounting. One subtlety that arises is how to best communicate a reward from the environment to the internal policy. Notice that in Algorithm 1, after the first internal action, r_t is set to 0. This has the effect of preserving the return and associated discounting schedule in the external environment. Although in the case of uninformative action binarization one can map a discounted external setup to an equivalent discounted internal setup (Majeed and Hutter 2020), attempting a more general construction along those lines in our case requires time-dependent, and worse, history-dependent discounting which runs into both technical and computational challenges.

7 Related Work

We now discuss some related work.

Compression-based RL. Using compression to aid with machine learning goes back to at least the work by Frank, Chui, and Witten (2000), where compression-based models were compared to classical machine learning methods on a number of natural language problems. More recently, Hamilton, Fard, and Pineau (2013) used compression with Predictive State Representation on domains with large observation spaces to aid with the intractability. Botvinick et al. (2015) discussed the internal representation of reinforcement learning, specifically a natural or efficient coding of the internal representation. Veness et al. (2015) used compression-based techniques for policy evaluation via action-value estimation.

Large transformer/language models used to aid RL. Language models have had a recent resurgence, starting with *Attention is all you need* (Vaswani et al. 2017) and being followed by the success of GPT-2 (Radford et al. 2019) and GPT-3 (Brown et al. 2020). Despite the accomplishment of language models on NLP tasks, there have only been a few circumstances where these techniques have translated to the field of reinforcement learning. Luketina et al. (2019) provides a survey of reinforcement learning methods which have been improved with the addition of natural language approaches. One such example is Kaplan, Sauer, and Sosa (2017), where natural language methods were used with deep reinforcement learning to play Atari games. In addition to this, recent work on using transformer models with reinforcement learning includes: Parisotto et al. (2020), Noever, Ciolino, and Kalin (2020) train GPT-2 (Radford et al. 2019) on the PGN format to learn chess, Ciolino, Kalin, and Noever (2020) trained GPT-2 in a similar way to learn Go, and Stein, Filchenkov, and Asadulaev (2020) used Transformers for Deep Q-learning to play Atari games. Krause et al. (2020) introduced a coding scheme to improve small language models.

8 Future Work

Arithmetic encoding has a number of extensions which deserve further investigation in the context of reinforcement learning. In particular, one can generalise arithmetic encoding to time-adaptive coding distributions, which is known as adaptive arithmetic encoding. While one could crudely incorporate this notion into our existing work, a more complete treatment would require going outside the MDP formalism.

Finally from a theory perspective, there are some additional generalizations that can be made, though they are beyond the scope of this paper. These include a more thorough treatment of the infinite horizon case by using discounting, investigating setups which do not require an end of action

symbol $\top$ (as discussed earlier) and allowing the external action space to be countably infinite.

9 Conclusion

In this work we have laid the conceptual foundations for information-theoretic actuation. We revisited the meaning of action in reinforcement learning, and explored a particular type of internal viewpoint. We have demonstrated how our method is theoretically well justified, by formally connecting it to a traditional reinforcement learning MDP setup. We argued that such a framework is well positioned to take advantage of the recent progress in large sequence/language models for multitask reinforcement learning problems over large action spaces. The next step is to explore application of this formalism in conjunction with modern sequence modelling techniques on some benchmark problems to better understand the potentials and limitations of this approach. Multi-task RL problems with vastly different action spaces seem the most natural setting where our approach could have immediate impact.

A Acknowledgements

We thank Csaba Szepesvari, Nando de Freitas, Oriol Vinyals, and Pedro Ortega for some helpful discussions. This work has been supported in parts by the Australian Research Council under grant DP150104590.

References

Bertsekas, D. P.; and Tsitsiklis, J. N. 1996. *Neuro-dynamic programming*. Athena Scientific.

Botvinick, M.; Weinstein, A.; Solway, A.; and Barto, A. 2015. Reinforcement learning, efficient coding, and the statistics of natural tasks. *Current opinion in behavioral sciences*, 5: 71–77.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Ciolino, M.; Kalin, J.; and Noever, D. 2020. The Go Transformer: Natural Language Modeling for Game Play. In *2020 Third International Conference on Artificial Intelligence for Industries (AI4I)*, 23–26. IEEE.

Cover, T. M. 1999. *Elements of information theory*. John Wiley & Sons.

Ecoffet, A.; Huizinga, J.; Lehman, J.; Stanley, K. O.; and Clune, J. 2021. Go-Explore: a New Approach for Hard-Exploration Problems. arXiv:1901.10995.

Frank, E.; Chui, C.; and Witten, I. H. 2000. Text categorization using compression models.

Hamilton, W. L.; Fard, M. M.; and Pineau, J. 2013. Modelling sparse dynamical systems with compressed predictive state representations. In *International Conference on Machine Learning*, 178–186. PMLR.

Janner, M.; Li, Q.; and Levine, S. 2021. Reinforcement Learning as One Big Sequence Modeling Problem. arXiv:2106.02039.

Kaplan, R.; Sauer, C.; and Sosa, A. 2017. Beating atari with natural language guided reinforcement learning. *arXiv preprint arXiv:1704.05539*.

Krause, B.; Gotmare, A. D.; McCann, B.; Keskar, N. S.; Joty, S.; Socher, R.; and Rajani, N. F. 2020. Gedi: Generative discriminator guided sequence generation. *arXiv preprint arXiv:2009.06367*.

Küttler, H.; Nardelli, N.; Miller, A. H.; Raileanu, R.; Selvatici, M.; Grefenstette, E.; and Rocktäschel, T. 2020. The nethack learning environment. *arXiv preprint arXiv:2006.13760*.

Luketina, J.; Nardelli, N.; Farquhar, G.; Foerster, J.; Andreas, J.; Grefenstette, E.; Whiteson, S.; and Rocktäschel, T. 2019. A survey of reinforcement learning informed by natural language. *arXiv preprint arXiv:1906.03926*.

MacKay, D. J. C. 2003. *Information Theory, Inference, and Learning Algorithms*. Copyright Cambridge University Press.

Majeed, S. J.; and Hutter, M. 2020. Exact Reduction of Huge Action Spaces in General Reinforcement Learning. *arXiv preprint arXiv:2012.10200*.

Milan, K.; Veness, J.; Kirkpatrick, J.; Bowling, M.; Koop, A.; and Hassabis, D. 2016. The Forget-me-not Process. In Lee, D.; Sugiyama, M.; Luxburg, U.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533.

Noever, D.; Ciolino, M.; and Kalin, J. 2020. The Chess Transformer: Mastering Play using Generative Language Models. *arXiv preprint arXiv:2008.04057*.

Orseau, L.; and Lelis, L. H. 2021. Policy-Guided Heuristic Search with Guarantees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 12382–12390.

Orseau, L.; Lelis, L. H.; Lattimore, T.; and Weber, T. 2018. Single-agent policy tree search with guarantees. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 3205–3215.

Ortega, P. A.; Wang, J. X.; Rowland, M.; Genewein, T.; Kurth-Nelson, Z.; Pascanu, R.; Heess, N.; Veness, J.; Pritzel, A.; Sprechmann, P.; Jayakumar, S. M.; McGrath, T.; Miller, K.; Azar, M.; Osband, I.; Rabinowitz, N.; György, A.; Chiappa, S.; Osindero, S.; Teh, Y. W.; van Hasselt, H.; de Freitas, N.; Botvinick, M.; and Legg, S. 2019. Meta-learning of Sequential Strategies. arXiv:1905.03030.

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

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8): 9.

Rissanen, J.; and Langdon, G. G. 1979. Arithmetic coding. *IBM Journal of research and development*, 23(2): 149–162.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Stein, G.; Filchenkov, A.; and Asadulaev, A. 2020. Stabilizing Transformer-Based Action Sequence Generation For Q-Learning. *arXiv preprint arXiv:2010.12698*.

Strehl, A.; Li, L.; and Littman, M. 2009. Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research*, 10: 2413–2444.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Szepesvári, C. 2010. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1): 1–103.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Veness, J.; Bellemare, M.; Hutter, M.; Chua, A.; and Desjardins, G. 2015. Compress and control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.

Veness, J.; Lattimore, T.; Budden, D.; Bhoopchand, A.; Mattern, C.; Grabska-Barwinska, A.; Sezener, E.; Wang, J.; Toth, P.; Schmitt, S.; et al. 2019. Gated linear networks. *arXiv preprint arXiv:1910.01526*.

Witten, I. H.; Neal, R. M.; and Cleary, J. G. 1987. Arithmetic coding for data compression. *Communications of the ACM*, 30(6): 520–540.

Supplementary Material

Proof of Theorem 2

Proof. Instead of expanding the entire action-value function we will show that a single interaction between μ and Π are equal to an interaction between ϑ and π. Let s', r, a' be arbitrary states, rewards and actions, then

$$\begin{aligned}
&\mu(s'r|s,a)\Pi(a'|s') \\
&\overset{(a)}{=} \mu(s'r|s,a) \sum_{\substack{q'\in\mathbb{D}_{s'}:\\ a'=\tau(D_\rho(q'|s'))}} \prod_{i=1}^{|q'|} \pi(q'_i|s'q'_{<i}) \\
&\overset{(b)}{=} \vartheta(s'\epsilon r|sq,b) \sum_{\substack{q'\in\mathbb{D}_{s'}:\\ a'=\tau(D_\rho(q'|s'))}} \prod_{i=1}^{|q'|} \pi(q'_i|s'q'_{<i}) \\
&\overset{(c)}{=} \sum_{\substack{q'\in\mathbb{D}_{s'}:\\ a'=\tau(D_\rho(q'|s'))}} \vartheta(s'\epsilon r|sq,b)\pi(q'_1|s'\epsilon)\times \\
&\qquad \prod_{i=2}^{|q'|} \vartheta(s'q'_{<i}0|s'q'_{<i-1},q'_{i-1})\pi(q'_i|s'q'_{<i}).
\end{aligned}$$

Step (a) comes from Equation 1; step (b) comes from the first case of Definition 1; step (c) comes from the second case of Definition 1, where $\vartheta(s'q'_{<i}0|s'q'_{<i-1},q'_{i-1}) = 1$.

Therefore in the expansion of the action-value function we can replace one with the other,

$$\begin{aligned}
Q^\Pi_\mu(s,a) &= \mathbb{E}^\Pi_\mu\left[\sum_{t=1}^m r_t|s,a\right] \\
&= \sum_{s_1,r_1,a_1} \mu(s_1r_1|s,a)\Pi(a_1|s_1)\dots \sum_{s_m,r_m,a_m} \mu(s_mr_m|s_{m-1},a_{m-1})\Pi(a_m|s_m)\sum_{t=1}^m r_t \\
&= \sum_{s_1,r_1,a_1}\left(\sum_{\substack{q'\in\mathbb{D}_{s_1}:\\ a_1=\tau(D_\rho(q'|s_1))}} \vartheta(s_1\epsilon r_1|sq,b)\pi(q'_1|s_1\epsilon)\prod_{i=2}^{|q'|}\vartheta(s_1q'_{<i}0|s_1q'_{<i-1},q'_{i-1})\pi(q'_i|s_1q'_{<i})\right) \\
&\dots \sum_{s_m,r_m,a_m}\left(\sum_{\substack{q'\in\mathbb{D}_{s_m}:\\ a_m=\tau(D_\rho(q'|s_m))}} \vartheta(s_m\epsilon r_1|s_{m-1}\epsilon,b)\pi(q'_1|s_m\epsilon)\prod_{i=2}^{|q'|}\vartheta(s_mq'_{<i}0|s_mq'_{<i-1},q'_{i-1})\pi(q'_i|s_mq'_{<i})\right) \\
&\sum_{t=1}^m r_t \\
&= \mathbb{E}^\pi_\vartheta\left[\sum_{t=1}^m r_t|sq,b\right] = Q^\pi_\vartheta(sq,b)
\end{aligned}$$

Notice that the horizon is defined externally, i.e. an m-horizon return in the external environment corresponds to m decoded external actions, and not the number of internal actions taken. □

Proof of Proposition 3

Proof. ($\Rightarrow$)

Let $\mathcal{N} := \{x_{1:\infty} : \theta(x_{1:n}) \not\to \theta_0\}$ be the set of sequences on which convergence fails. The consistency assumption implies

$$\begin{aligned} 0 &= P_{\theta_0}[\mathcal{N}] \\ &= \sum_{x_{1:k} \in \mathcal{X}^k} P_{\theta_0}[\mathcal{N}|x_{1:k}] P_{\theta_0}[x_{1:k}] \\ &\geq P_{\theta_0}[\mathcal{N}|x_{1:k}] P_{\theta_0}[x_{1:k}]. \end{aligned}$$

Since $P_{\theta_0}[x_{1:k}] > 0$ by assumption, the RHS can only be 0 if $P_{\theta_0}[\mathcal{N}|x_{1:k}] = 0$, which implies $P_{\theta_0}[x_{1:k} \times \mathcal{X}^\infty \setminus \mathcal{N}|x_{1:k}] = 1$, hence $\theta(x_{1:k}X_{k+1:n}) \to \theta_0$ with $P_{\theta_0}[\cdot|x_{1:k}]$-probability 1.

($\Leftarrow$)

Again let $\mathcal{N} := \{x_{1:\infty} : \theta(X_{1:n}) \not\to \theta_0\}$ be the set of sequences on which convergence fails. The consistency assumption implies that for all $x_{1:k}$ for which $P_{\theta_0}[x_{1:k}] > 0$, we have $P_{\theta_0}[x_{1:k} \times \mathcal{X}^\infty \setminus \mathcal{N}|x_{1:k}] = 1$, which implies that $P_{\theta_0}[\mathcal{N}|x_{1:k}] = 0$. Therefore,

$$P_{\theta_0}[\mathcal{N}] = \sum_{x_{1:k} \in \mathcal{X}^k : P_{\theta_0}[x_{1:k}] > 0} P_{\theta_0}[\mathcal{N}|x_{1:k}] P_{\theta_0}[x_{1:k}] = 0$$

which implies $P_{\theta_0}[\mathcal{X}^\infty \setminus \mathcal{N}] = 1$, hence $\theta(X_{1:n}) \to \theta_0$ with P_{θ_0}-probability 1. $\square$