RECONFIGURABLE REINFORCEMENT LEARNING NETWORKS

In humans, the process of learning is not only driven by the environment and structure of the brain. The development of the brain-structure itself defines what learning may take place; thus the conditions and patterns which direct brain formation are primary and total for the success of learning. As artifical intelligence research continually generates and publishes on novel structures discovered by humans, this work is centered around how the novel structures can be discovered using RLNs. This subject is frequently included in the subject of general intelligence, and is famous for both its philosopical and computational complexity, as well as its difficulty in finding funding. There have been previous works on this subject, such as [Consciousness as a State of Matter] [] [].

Specifically, this work presents a unification method for online learning (via Reinforcement Learning) and offline learning (via Backpropagation). In the most general sense, this work demonstrates an apporach to the self-structuring of parametric models. First, it is reviewed that Concurrent Markov Decision Processes (CMDPs) can model parametric structure and facilitate optimal behaviour even when subject to large state spaces and generous state uncertainty. Second, it is shown that a variation of CMDPs called Reconfigurable Learning Networks (RLNs) can learn parametric decision networks. RLNs in structure and behaviour turn out to be equilivent to the structure and behaviour of feed-forward neural networks. Lastly, a few empirical examples are demonstrated, beginning with the MINST dataset. Two main contributions are made: First, RLNs can be trained online and offline, using Reinforcement Learning and then Backpropagation; online learning stimulates network growth and adaptation immediately, whereas backpropagation seems to be an ideal phase for network pruning. Second, an RLN can achie e empirical success even when the reward function for the system is changed dynamically. Thus both a degree of empirical success and general learning have been achieved.

In order for a generally intelligent system to operate, solutions to several open problems need to be solved anaytically and/or heuristically. In this work, we present the related problem categories in the Introduction (Section 1), and include background on each area. Second, most of this work is focused around the reconfiguration of existing MDP models, so Section 2, Mapping, includes work on transfer learning and analytical anaysis. Third, we express how convergence of behaviour policies can be preserved despite online RLN restructuring (Section 3). The tradeoff between network structure and computation time in learning is expressed analytically (Section 4). Lastly, it is shown that RLNs are actually just feed-forward Neural Networks, which adds the ability to use back propagation and other techniques on discovered models (Section 5).

In this work due to the diffiucty of the subject matter initally, models are assumed noiseless and stochastically stable. It is expected that later work will broaden this work by considering state uncertainty, and non-stationary problems.

NOTATION

In general, most online optimization problems can be expressed as fully observable Markov Decision Processes (MDPs) as $\langle S, A, T, R, \pi \rangle$ tuples:

- $S \subseteq R^n$: A discrete collection of states.

- $A \subseteq R^n$: A discrete collection of actions.
- $T(s'|a,s) \in R$: A stably stochastic transition function, where $\sum_{s \in S} T(s|a,s') = 1$
- $R(s'|a,s) \in R$: A stable stochastic reward fuction
- $\pi : S \times A \to R$: A non-negative behaviour policy with the general property, $\sum_{a \in A} \pi(s,a) = 1$

In general, we can express behaviour in this domain as a policy $\pi : S \to R$ **[? looked like this, but would $\pi : S \to A$ make more sense?]**. Particular attention is given to the optimal strategy.

In prior work the issue of tractability and subsequent decomposition have been articulated. In this work the subject of learning and generalizing this decomposition work into a General framework is discussed.

Ⓐ Theory
- Section 1: Background & Introduction — Background of relevant research & RLN introduction
- Section 2: Mapping — a generalized set of mapping & deconstruction operations (parent, ch
- Section 3: Convergence (16): — parent, child, reward optimization, complexity
- Section 4: Worst Case Performance (23): — system behaviour with malformed problems

Ⓑ NN paper
- Section 5: Neural Networks (24): — RRLN are just feed forward Neural Networks
- $\hookrightarrow$ (N.)

Special topics:

Temporal Difference (A1): how to discover & change time basis/scale

Transitional Learning (A2): how to re-use and generalize transitional models

Financial Systems (A3): how to use with financial systems

Origins (E1-E4): original examples and sketches

Transitional Encoding (E5-E6): Continuous Gaussian mixture models & applications

1 Background and Introduction

Introduction: A reconfigurable reinforcement learning method

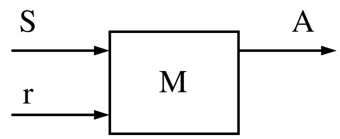
1.1 RLN MDP structure

The general approach that is taken to form an RLN is "split" one single MDP into parent and child processes. Doing so assumes the two process are partly independent [CMDP].

1. Largely, the models may be independent.
2. The child and parent may include elements of each other's MDP definition in their own definition.

This section focuses on framing a model. In Sections 2 and later, subjects related to behavior optimality, convergence, and computational complexity are considered.

In order to consider the formation of a reconfigurable reinforcement network, it is required to analytically group all aspects of a process and a behavior policy into one tuples.



To do this, assume a Markov decision process M which can be internally modeled as a tuple $M = \langle S, A, T, R, \pi, \tilde{T}, \tilde{R} \rangle$

S – a set of states $s \in S$ which may be experienced by M

A – a set of actions $a \in A$ that may be executed

T – a true transitional probability, $T(s'|a, s)$ expressing the probability of executing an action a in state s before ending up in later state s'.

R – is a reward function which quantifies how desirable a transition $R(s'|a, s)$ is. $R : S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$

$\tilde{T}$ – is the current model of T. The goal of $\tilde{T}$ is thus $\tilde{T} \sim T$

$\tilde{R}$ – is the predicted reward of the system, constructed from observation of R, s.t. $\tilde{R} \rightarrow R$.

π – is an action selection policy, ideally chosen to maximize expected reward, an optimal policy is denoted π^*.

Ideally

$$\pi^*(s) = \arg\max_a \sum_{s'} \underbrace{R(s'|a, s)T(s'|a, s) + \gamma V(s')}_{\text{expected reward}}$$

and

$$\tilde{\pi}(s) = \arg\max_a \sum_{s'} \tilde{R}(s'|a, s)\tilde{T}(s'|a, s) + \gamma \tilde{V}(s')$$

[Note that $\tilde{V}$ hasn't been defined in the previous equation.]

ENCODING

$$\pi^*(s) = \arg\max_a \sum_{s'} R(s'|s, a)T(s'|s, a) + \gamma V(s')$$

where

$$V(s) = \sum_{s'} R(s'|s, a)T(s'|s, a) + V(s') \,.$$

A bellman backup can be used [Bellman backup]. In online applications stochastic gradient descent can be applied to regress to locally optimal solutions. This allows estimation of optimal policy

$$\pi^*(s) = \arg\max_a Q(s,a)\,.$$

To encode the expected reward over all states, typically Q-values are kept: $Q(s,a) \sim \sum R(s'|a,s)T(s'|a,s) + \gamma V(s')$ and $Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha\left(R(s'|a,s) - Q_t(s,a) + \gamma \arg\max_{a'} Q(s',a')\right)$.
To render the process M separable, it is necessary to decouple the transitional values T from the reward values R. Thus, to directly encode $Q(s,a)$ using π is prohibitive.
knowing:

$$Q : S \times A \to \mathbb{R}_{\geq 0}$$

indirect encoding:

$$\pi : \{S \times A \times S \times \mathbb{N} | R\} \to Q$$

If the definitions of S or A change, then Q must be reinitialized. Alternatively, $\tilde{T}$ and $\tilde{R}$ are defined as intermediate encoding functions. Thus we define

$$\pi : \{S \times A \times S \times \mathbb{N}\} \to \tilde{T}, \tilde{R}, Q$$

simple transition

$$\tilde{T}_{t+1}(s'|s,a) = \frac{\text{freq}(s'|s,a)}{\text{freq}(s,a)}$$

simple reward

$$\tilde{R}_{t+1}(s'|s,a) = \tilde{R}(s'|s,a) + \alpha_R\left(\tilde{R}(s'|s,a) - R(s'|s,a)\right)$$

$$f_Q : \tilde{T}_t \times \tilde{R}_t \to Q_t$$

In this paper we rely on a method of extracting dynamic Q-values from an encoded transition and reward function $(\tilde{T}, \tilde{R})$. The motivation for this encoding is that it allows mapping the transition function into multiple spaces, and allows the reward function to be altered. The significance of this finding is covered in **???** Price wash **???**.

1.2 Reconfiguration of M

Reconfiguring a process M allows some intractable MDPs to be rendered tractable. As an example, take an MDP M modeling a 3-dimensional foraging experiment with three thousand positions on the x, y, and z axes respectively. This process will consume over three billion memory locations and may be impossible to explore. If this system is broken into three sub problems, each targeting a special axis, then only nine thousand memory locations need be consumed. This decreases memory requirements by an exponential factor.
This section presents a method of decomposition that, when followed, introduces no degeneration of the regressed policy $\pi^*(s,a)$. The summary of these conditions is presented. There is no free lunch: the trade-off for saving in space is exponentially increased computational burden. As in all problems, finding the balance between space and computational complexity is required.

1.2.1 Introduction to approach

The general approach is related to factor analysis, or clustering, eigenvalue decomposition, or projected component analysis, with which the reader may be familiar. The process is split such that a subspace $S_i \subseteq S$, and action space $A_i \subseteq A$ seem independent of subspaces S_k, A_k, for the purposes of generating reward $R(s'|a,s)$. Specifically, it may be observed that the reward is homogeneous, such that

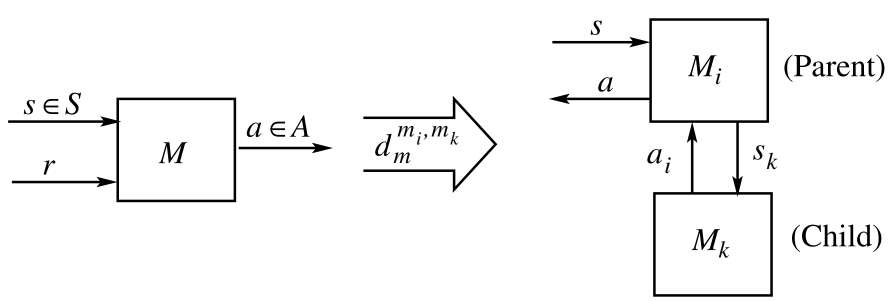
$$\forall k_2, a_2 : R\left(S'_i \cup S'_{k_1} \middle| S_i \cup S_{k_1}, a_i \cup a_{k_1}\right) \sim R\left(S'_i \cup S'_{k_2} \middle| S_i \cup S_{k_2}, a_i \cup a_{k_2}\right) \ .$$

In this case, it is likely that the problem may be separated. Even in the case where variance of reward is high, convergence is expected.

Splitting is done using a decomposition function

$$d_M^{M_i,M_k} = M \longrightarrow \left\{ M_i, M_k \middle| \begin{array}{l} S_i \times (S_k/\{s_i\}) = S, S_k \times (S_i/\{s_k\}) = S \\ A = A_i \cup A_k \\ \tilde{T} \sim d^{-1}(d(\tilde{T})), d(\tilde{T}) = \tilde{T}_i, \tilde{T}_k \\ \tilde{R} \sim d^{-1}(d(\tilde{R})), d(\tilde{R}) = \tilde{R}_i, \tilde{R}_k \end{array} \right\}$$

where d represents belief mapping functions that decompose and recompose $\tilde{T}$ and $\tilde{R}$. This allows M to be mapped as new spaces and observations are encountered. The decomposition process breaks one MDP into a parent and child:



Although many decomposition strategies are possible, this work presents a specific approach related to parent–child decomposition. This specific decomposition allows for assured policy convergence (see Section **??**)

Definitions

$\underline{M}$ $S = (S_i/S_k) \times (S_k/S_i)$

$A = A_i \cup A_k$

$T = P(S \times A \times S)$

$R =$ real, positive, convergent stochastic as $t \to \infty$

$$R(s', a, s) = R\begin{pmatrix} s'_i & & a_i & & s_i \\ & , & & , & \\ s'_k & & a_k & & s_k \end{pmatrix}$$

The system can be broken into the following MDP definitions

M_i – Parent

S_i – a collection of states, $s_i \in S_i$

A_i – a collection of actions, $a_i \in A_i$

$\tilde{T}(s'_i|s_i, a_i)$ – the observed probability of executing action a_i in state s_i and ending up in state s'_i

$$\left.\begin{matrix} \tilde{T} \\ \tilde{R} \end{matrix}\right\} \text{Covered Pages on BII p12-14}$$

$P(s'_i|s_i, a_i)$ is observed directly and is $\tilde{T}(s'_i|s_i, a_i)$

$$\tilde{R}_t\left(\begin{matrix} s'_i \\ a'_k \end{matrix}\,\middle|\, \begin{matrix} a_i \end{matrix}\;,\;\begin{matrix} s_i \\ a_k \end{matrix}\right) = \tilde{R}_t\left(\begin{matrix} s'_i \\ s'_k \end{matrix}\;,\;\middle|\;\begin{matrix} a_i \\ a_k \end{matrix}\;,\;\begin{matrix} s_i \\ s_k \end{matrix}\right) = \tilde{R}(s'|a, s)$$

S, T_i, S_k, S'_k are not directly observable by process M_i, by design. Importantly, some facts are known about a_k and a'_k which will be exploited in Section **??**.

ii) $a_k = \pi_k(s_k)$

iii) $a'_k = \pi_k(s_k)$

iiii) (s_k, s'_k) chosen indirectly by $\pi_k(\cdot)$ in a manner that assuming monotonic increase in reward,

$$\text{as } t \to \infty \quad E[R_{t+1}(\cdot)] \geq E[R_t(\cdot)]\,.$$

Local convergence of this process on a behavior policy $\pi^*_i(\cdot)$ is assured.

M_k – child

S'_i – all child states, $s_k \in S_k$

$a_k \in A_k$

$\tilde{T}(s'_k|s_k, a_k)$

$R_t(s'_k, a_k, s_k) = R_t\left(\begin{smallmatrix} s_i \\ s'_k \end{smallmatrix}, \begin{smallmatrix} a_i \\ a_k \end{smallmatrix}, \begin{smallmatrix} s_i \\ s_k \end{smallmatrix}\right)$ s.t. s_i, s'_i are chosen by another process, and

$$\boxed{A^*} \quad \longrightarrow \quad E[R_{t+1}(\cdot)] \geq E[R_t(\cdot)]$$

It is direct to note that both processes M_i and M_k are guaranteed to converge on locally optimal policies $\pi_i(\cdot)$, $\pi_k(\cdot)$ as the limit of time approaches infinity.

In following sections next steps are discussed:

- $\pi_i^* \times \pi_k^* \to \pi^*$ utilizing the degenerate-optimal policies (Section **??**)
- $d_M^{M_i, M_k}$ – how to map values while preserving information, avoiding degeneracies (Section 2)

2 MAPPING FUNCTION

After defining Parent and Child MDPs, it is important to note how to map information from a centralized MDP onto a parent-child pair: (equation $d_{M_i}^{M_j,M_k}$) $d_{M_i}^{M_j,M_k}$. To expedite convergence and computational proofs that follow, a mapping function $d_{M_i}^{M_j,M_k}$ is assume to have Basic Requirements (Sec. 2.1). Afterward the process is explained (Sec. 2.2).

2.1 Basic mapping requirements

S_i, S_j, S_k: $\quad S_j \times (S_k/S_j) \supseteq S_i, \quad S_k \times (S_j/S_k) \supseteq S_i$

Sets S_k and S_j must be able to be combined to make the original set.

A_i, A_j, A_k: $\quad A_i \subseteq A_j \cup A_k$

Also, action sets must combine to recover the original action set.

T_i, R_i $\sim$ unknown/unknowable, stable decomposition

The true transition and reward functions may remain unknown.

more$\longrightarrow$ $\boxed{\begin{array}{c}\text{assumed} \\ * \text{ important to select so that } \tilde{T}_i \text{ \& } \tilde{T}_k \text{ seem independent}\end{array}}$

2.2 Mapping process

Transition mapping:

$\exists f_1 : \tilde{T}_i \to \tilde{T}_j, \tilde{T}_k$, invertible; $\tilde{T}_i = f_1\left(f^{-1}\left(\tilde{T}_i\right)\right)$
$\exists f_2 : \tilde{R}_i \to \tilde{R}_j, \tilde{R}_k$, invertible; $\tilde{R}_i = f_2\left(f_2^{-1}\left(\tilde{R}_i\right)\right)$

<u>Transition function mapping: knowing $s_x \in S_x$, $s_y \in S_y$</u> (1 way)

Goal $\exists f : P(S_x|A, S_x) \leftarrow P(S|A, S)$
knowing $P(S_x \times S_y|A_x \cup A_y, S_x \times S_y) = P(S|A, S)$
Clearly:

$$P(s'_x|s_x, a_x) = \sum_{s'_y}\sum_{a_y}\sum_{s_y} P\left(s'_x, s'_y|s_x, s_y, a_x, a_y\right) P\left(a_y|S_y\right) P\left(s_y\right)$$

$$\therefore \quad \tilde{T}\left(s'_x|s_x, a_x\right) = \sum_{s'_y}\sum_{a_y}\sum_{s_y} \tilde{T}\left(s'|a, s\right) \quad \underbrace{\tilde{\pi}\left(a_y|s_y\right)}_{\text{require policy mapping}} \quad P(s_y)$$

Action mapping:

Action mapping (1 way)

Next, we can consider an action mapping where actions from A can be randomly assigned to A_x, A_y: $A_x \leftarrow \{a \in A' | A' \subseteq A\}$, $A_x, A_y \subseteq A$, $A_x \cup A_y = A$, $A_x \neq \{\}$, $A_y \neq \{\}$.

General approach: High reward for **???**ve states

Given $\tau \in \mathbb{R}$, $\pi(a|s)$, $\tilde{Q}(a,s)$ then

$$A_x \leftarrow A_x \cup \left\{ a \middle| \underbrace{\pi(a|s)\tilde{Q}(a,s) > \tau}_{\text{condition}} \right\}$$

or, more usefully/generally

$$A_x \leftarrow A_x \cup \left\{ a \middle| \underbrace{\left(\pi\left(a \middle| S_s^{R'}\right) \tilde{Q}(a, S^{*\prime}) > \tau\right.}_{\text{condition}} \right\}$$

where

$$S_s^{*\prime} = \{S' | S/s_s^* \neq S\} \qquad \text{(see p. ???)}$$

Condition options:

$*$reformulation over set S vs. $s \in S$ $\quad \begin{cases} \text{a) } \pi(a|s)\tilde{Q}(a,s) > \tau & \cdots \text{ High reward} \\ \text{b) } \pi(a|s)\tilde{Q}(a,s) > \tau, \quad \pi(a|s) > 0 & \cdots \text{ small reward} \end{cases}$

(Network approach)

3 Mapping as a process

In this section it is described how, and when, the function $d_{M_i}^{M_j,M_k}$ is applied. In fact, instead of structing the application of $d_{M_i}^{M_j,M_k}$ as human chosen conditional statements, it is possible to consider when to "split" and "merge" and MDP M_i into M_j, M_k as a learning problem. This problem is expressed as the mapping problem, and modelled as a learning problem MDP-map (M_{map}):

$$M_{\text{map}}^i = \left\langle S_{\text{map}}^i, A_{\text{map}}^i, T_{\text{map}}^i, R_{\text{map}}^i \right\rangle$$

$$S_{\text{map}} = f\left(d\left(M_i, \mathcal{P}\left(M_j\right), P\left(M_k\right)\right)\right)\,, \quad \text{s.t. } f: d_{M_i}^{M_j,M_k} \to \mathbb{R}$$

where $d\left(M_i, M_j, M_k\right) = d_{M_i}^{M_j,M_k}$.

The state encodes all possible configurations for the MDP split:

[FIGURE]

$$A_{\text{map}}^i = \text{a set of actions formed from the the decomposition and recomposition functions}$$

$$A_{\text{decomposition}} \cup \{a_r\}$$

where a_r is a special recomposition function.

$$T_{\text{map}}^i = \begin{cases} 1\,, & (S = S_1 \text{ and } a \neq a_r) \text{ or } (S \neq S_1 \text{ and } a = a_r) \\ 0\,, & \text{otherwise} \end{cases}$$

$$R_{\text{map}}^i = \sum_{e=\text{epoch}} R^i(e)$$

Given $(S_{\text{map}}, A_{\text{map}}, T_{\text{map}}, R_{\text{map}}^i, \pi_{\text{map}})$, applied to $M = \langle S_x, A_x, \tilde{T}_x, R_x, \tilde{\pi}_x \rangle$, we may trivially define $M_y = \langle S_y, A_y, \tilde{T}_y, R_y, \pi_y \rangle$ in a method consistent with Bush, p. 74, with M_x being the parent process and M_y being the child.

a) for R_{map}^i, there are five versions $i \in \{1, \ldots, 5\}$

b) Given M_x, M_y, a merge is also possible, so recovering M

c) we can perform temporal Sink actions on an MDP (Book I, p. 88)

  $\hookrightarrow$ reduce resolution

  $\hookrightarrow$ re-increase resolution

<u>Actions</u>

$\therefore$ Seven "actions" can be performed on an MDP: (M_R)

<u>State</u>

$$\left\{R_{\text{map}}^{i}\right\}_{i=0}^{5} \cup \{\text{merge}\} \times \left\{\text{scale up}^{i}\right\}_{i=0}^{5} \cup \{\text{normal}\} \cup \{\varnothing\}$$

<u>Reward</u>

$$R(s', a, s) = \sum_{l \in e} R(l) \qquad \text{reward during a trajectory}$$

$e =$ epoch

<u>Transition</u>

-easy to explain in MS Word

$$T = \begin{cases} 1 \text{ – allow } \textbf{???} \\ 0 \text{ – otherwise} \end{cases}$$

Reward function mapping (5 way)

knowing $R\left(\{s_x, s_y\}|a, \{s'_x, s'_y\}\right) = R(s|a, s)$

Mapping Policy (1 way)

finding: $f\tilde{\pi}(s) \rightarrow f\tilde{\pi}(s_y)$

$$\tilde{\pi}(a_y|s_y) \leftarrow \sum_{s_x} \sum_{a_x} \tilde{\pi}\left(\{a_x, a_y\}|\{s_x, s_y\}\right) P(s_x)$$

3.1

3.1.1 Proof of Decomposition Correctness (Process 1)

The following section demonstrates

MDP Policy Decomposition

First, it is true that the optimal policy $\pi^*(s)$ for a centralized MDP will arise from maximizing the expected reward.

$$\pi^*(s) = \arg\max_a \sum_{s'} R(s,a,s')P(s'|s,a) + \gamma V(s')$$

It is now shown that a set of MDPs (M_i, M_k) can be observed to create an optimal policy $\exists f s.t.\ f\left(\pi_i^*, \pi_k^*\right) \sim \pi^*$. Given $\pi_i^*(S_i, A_k)$, $\pi_k^*(S_k)$

$$1.\quad \pi^*(s_i, s_k) = \arg\max_{a_i, a_k} \sum_{s_i'} \sum_{s_k'} R\left((s_i, s_k), (a_i, a_k), (s_i', s_k')\right) P\left((s_i', s_k')|(s_i, s_k), (a_i, a_k)\right)$$

- Lemma 1 – augmentation with a_k where $a_k' = \pi^*(s_k')$

$$2.\quad \pi^*(s_i, s_k) = \arg\max_{a_i, a_k} \sum_{s_i'} \sum_{s_k'} R\left((s_i, s_k, a_k), (a_i, a_k), (s_i', s_k', a_k')\right) P\left((s_i', s_k', a_k')|(s_i, s_k), (a_i, a_k)\right)$$

- Lemma 2 – Simplification

$$* \quad \text{assume} \quad \overbrace{\arg\max_{a_i, a_k} \equiv \arg\max_{a_i} \arg\max_{a_k}}^{\text{separability}}$$

$$3.\quad \pi^*(s_i, s_k) = \arg\max_{a_i, a_k} \sum_{s_i'} R\left((s_i, a_k), (a_i, a_k), (s_i', a_k')\right) P\left(s_i', a_k'|(a_i, a_k), (s_i, s_k)\right)$$

Lemma 3

- separation of a_k, and $a_k \leftarrow \pi_k^*(s_k)$

$$4.\quad \pi^*(s_i, s_k) = \left(\arg\max_{a_i} \sum_{s_i'} R\left((s_i, a_k), a_i, (s_i', a_k')\right) P\left(s_i', a_k'|a_i, (s_i, s_k)\right)\right)$$

- Lemma 4

$$a_i = \pi_i^*(s_i) \longrightarrow^{\cup} \left(\arg\max_{a_k} \sum_{s_k'} R\left(s_k, a_k, s_k', a_k'\right) P\left(s_k'|a_k, s_k\right)\right)$$

$$\boxed{5. \quad \pi^*(s_i, s_k) = \pi_i^*(s_i, a_k) \cup \pi^*(s_k)}$$

Thus, given π_i^* and π_k^i, it is possible to infer $\pi^*(s_i, s_k)$.

We now prove lemmas involved.

Lemma 1:

Lemma 2:

Lemma 3:

Lemma 4:

3.1.2 Proof of Decomposition Correctness (Process #2)

MDP Policy Decomposition

$$\pi^*(s) = \arg\max_a \sum_{s'} R(s,a,s')P(s'|s,a) + \gamma V(s)$$

Given $\pi_i^*(S_i, A_k)$, $\pi^*(S_k)$

$$1. \quad \pi^*(s_i, s_k) = \arg\max_{a_i, a_k} \sum_{s_i'} \sum_{s_k'} R\left((s_i, s_k), (a_i, a_k), (s_i', s_k')\right) P\left((s_i', s_k')|(s_i, s_k), (a_i, a_k)\right)$$

Note: $\quad R\left((s_i, s_k, a_k), (a_i, a_k), (s_i', s_k')\right) \leftarrow R\left((s_i, s_k), (a_i, a_k), (s_i', s_k')\right)$

$\quad\quad R\left((s_i, s_k, a_k), (a_i, a_k), (s_i', s_k', a_k')\right) \leftarrow R\left((s_i, s_k), (a_i, a_k), (s_i', s_k')\right)$

$*$assume separability $\longrightarrow$

$$2. \quad \pi^*(s_i, s_k) = \arg\max_{a_i} \arg\max_{a_k} \sum_{s_i'} \sum_{s_k'} R\left((s_i, s_k, a_k), (a_i, a_k), (s_i', s_k', a_k')\right) P\left((s_i', s_k', a_k')|\cdot\right)$$

$$= \arg\max_{a_i} \sum_{s_i'} \sum_{s_k'} R\left((s_i, s_k, a_k), \begin{array}{c} a_i \\ a_k \end{array} \middle| \begin{array}{c} s_i' \\ s_k' \\ a_k' \end{array} \right) P\left(\begin{array}{c} s_i' \\ s_k' \\ a_k' \end{array} \middle| \begin{array}{c} a_i \\ a_k \end{array}, \begin{array}{c} s_i \\ s_k \\ a_k \end{array} \right)$$

$$\cup \arg\max_{a_k} \sum_{s_k'} R\left(\begin{array}{c} s_i \\ s_k \\ a_k \end{array}, \begin{array}{c} a_i \\ a_k \end{array} \middle| \begin{array}{c} s_i' \\ s_k' \\ a_k' \end{array} \right) P\left(\begin{array}{c} s_i' \\ s_k' \\ a_k' \end{array} \middle| \begin{array}{c} a_i \\ a_k \end{array}, \begin{array}{c} s_i \\ s_k \\ a_k \end{array} \right)$$

$$* \qquad \text{let } a_k^* = \arg\max_{a_k} \sum_{s_k'} R\left(\begin{array}{c} s_i \\ s_k \\ a_k \end{array}, \begin{array}{c} a_i \\ a_k \end{array} \middle| \begin{array}{c} s_i' \\ s_k' \\ a_k' \end{array} \right) P\left(\begin{array}{c} s_i' \\ s_k \\ a_k \end{array} \middle| \begin{array}{c} a_i \\ a_k \end{array}, \begin{array}{c} s_i \\ s_k \\ a_k \end{array} \right)$$

s. t. $s_i, s_i' \leftarrow \pi_i^*($

$a_i^* = \pi_i^*(s)$

$$3. \qquad = \arg\max_{a_i} \sum_{s_i'} R\left(\begin{array}{c} s_i \\ a_k \end{array}, \begin{array}{c} a_k \\ a_k^* \end{array}, \begin{array}{c} s_i' \\ a_k' \end{array} \right) P\left(\begin{array}{c} s_i' \\ a_k' \end{array} \middle| \begin{array}{c} a_i \\ a_k^* \end{array}, \begin{array}{c} s_i \\ a_k \end{array} \right) \cup \pi_k^*(s_k) = a_k$$

$$= \pi_i^*\left(s_i|\pi_k^*\right) \cup \pi_k^*(s_k)$$

4 CONVERGENCE

Policy convergence is considered for the parent and child learning process. Since both policies utilize information from external learning processess the proofs are non-trivial.

4.1 For the Parent MDP M_k

1. Assume the child's expected return has a monotic expected value.

$$E\left[R_t\left(s'_k|a_k, s_k\right)\right] \geq E\left[R_{t+1}\left(s'_k|a_k, s_k\right)\right]$$

2. Define reward as

$$R_t\left(s'_i|a'_i, s'_i\right) = R\left(\begin{array}{c|cc} s'_i & a_i & s_i \\ s'_k & a_k & s_k \end{array}\right)$$

i. $a_k = \pi_k(s_k)$

We de-

(s'_k, s_k) result from a_i s.t.

ii. $s'_k \sim T(S_k|\pi_i(s_k), s_k)$

iii. $s_k \sim T(S_k|\pi_i(s^*_k), s^*_k)$

fine the reward of the parent to be the reward received by the child process during observation (M_i observing M_k).

A) Parent is effective

$$0 > \min_{a_i} E\left[R\left(\begin{array}{c|cc} s'_i & a_i & s_i \\ s'_k & a^*_k & s_k \end{array}\right)\right] - E\left[R\left(\begin{array}{c|cc} s'_i & A_i & s_i \\ s'_k & A_k & s_k \end{array}\right)\right]$$

Child behaviour policy π_k is effective if the expected reward increases. In this case we consider a_k to be optimal.

B) Child is convergent

$$E\left[R_{t+1}\left(\cdot\left|\begin{array}{c} a_i \\ \pi^{\ k}_{t+1} \end{array}, \cdot\right.\right)\right] \geq E\left[R_{t+1}\left(\cdot\left|\begin{array}{c} a_i \\ \pi^{\ k}_{t+1} \end{array}, \cdot\right.\right)\right]$$

It is convergent if the centralized MDP's reward increases when observed.

assume some policy $\pi_k(\cdot)$ is both effective and convergent, then:

Ⓐ + Ⓑ → Ⓒ

C) The system must be convergent

- The parent chooses actions to maximize $R_t(A)$
- The child chooses actions to maximize $R_t(B)$

$$\boxed{*} \qquad \lim_{t\to\infty} R_t\left(s'_i|a_i, s_i\right) \sim R_{t+1}\left(s'_i|a_i, s_i\right)$$

4.2 For the child

$\boxed{*}$ trivial

1. Assume the child's expected return has a monotic expected value.

$$E\left[R_t\left(s'_k|a_k, s_k\right)\right] \geq E\left[R_{t+1}\left(s'_k|a_k, s_k\right)\right]$$

2. Define reward as

$$R_t\left(s'_i|a'_i, s'_i\right) = R\left(\begin{array}{c|cc} s'_i & a_i & s_i \\ s'_k & a_k & s_k \end{array}\right)$$

i. $a_k = \pi_k(s_k)$ We de-

(s'_k, s_k) result from a_i s.t.

ii. $s'_k \sim T(S_k|\pi_i(s_k), s_k)$

iii. $s_k \sim T(S_k|\pi_i(s^*_k), s^*_k)$

fine the reward of the parent to be the reward received by the child process during observation (M_i observing M_k).

A) Parent is effective

$$0 > \min_{a_i} E\left[R\left(\begin{array}{c|cc} s'_i & a_i & s_i \\ s'_k & a^*_k & s_k \end{array}\right)\right] - E\left[R\left(\begin{array}{c|cc} s'_i & A_i & s_i \\ s'_k & A_k & s_k \end{array}\right)\right]$$

Child behaviour policy π_k is effective if the expected reward increases. In this case we consider a_k to be optimal.

B) Child is convergent

$$E\left[R_{t+1}\left(\cdot\left|\begin{array}{c} a_i \\ \pi_{\substack{k \\ t+1}} \end{array}, \cdot\right.\right)\right] \geq E\left[R_{t+1}\left(\cdot\left|\begin{array}{c} a_i \\ \pi_{\substack{k \\ t+1}} \end{array}, \cdot\right.\right)\right]$$

It is convergent if the centralized MDP's reward increases when observed.

assume some policy $\pi_k(\cdot)$ is both effective and convergent, then:

Ⓐ + Ⓑ → Ⓒ

C) The system must be convergent

- The parent chooses actions to maximize $R_t(A)$
- The child chooses actions to maximize $R_t(B)$

$$\boxed{*} \qquad \lim_{t\to\infty} R_t\left(s'_i|a_i, s_i\right) \sim R_{t+1}\left(s'_i|a_i, s_i\right)$$

5 WORST CASE PERFORMANCE

5.1 Computational Complexity

Total Mapping

$$* A_R = \left\{ \begin{array}{l} \text{State 1, State 2, Action 1, Action 2} \\ \text{merge, time up, time down} \end{array} \right\}$$

Given $S \in \mathbb{R}^n$, define dimensions $\{i_s\}_{i_s=1}^n$
$A \in \mathbb{N}^m$, define dimensions $\{i_r\}_{i_r=1}^m$

Then, with an initial MDP $M = \langle S, A, T, R, \pi, M_R \rangle$, all possible "sub mdps" M_1, M_2, M_3, … represent the family of MDPs which can be created from M, $\mathcal{P}(M) = \{M_x | S_x \subseteq S, A_x \subseteq A, R, \pi \text{ from MDPs } \textbf{???}\}$ and each member M_x is characterized by a language $I_{sx} \subseteq \{i_s\}_{i_s=1}^n$ or $I_{sy} \subseteq \{i_r\}_{i_r=1}^m$ where $I_{sx} \times I_{sy}$ defines a space S_R, for the reconfiguration MDP to explore, with actions from A_R.

$$I \left| \begin{array}{l} \text{Reward is defined as average expected reward over an epoch } e. \\ \text{in terms of transition} \end{array} \right.$$

$$* S_R = \mathcal{P}\left(\{i_s\}_{i_s=1}^n\right) \times \mathcal{P}\left(\{i_r\}_{i_r=1}^m\right) \qquad \longleftarrow \text{ exponentional increase in space (stupid!)}$$

Problems

1) exponential space consumption
2) how to handle chaining/nesting
3) how to structure action choice policy

6 NN

① Summarize Regression ($y = mx + b$)

points $\{1, 2, 3, \ldots\}$

- easiest starting point is choosing "the best line"
- need a metric to define this idea of "best" mathematically
- describe the line: $f(x_1) = m(x_1) + b$

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

idea — choose line closest to points by taking sum of differences,

$$\text{minimize} \sum_i e_i\,, \qquad e_i = y_i - f(x_i) = y_i - m_i x_i + b m_0 x_0\,, \qquad x_0 = 1 \quad \text{(by convention)}$$

- apply square to remove negative/positive problem

$$e_i = (y_i - m_i x_i + b)^2$$

$$e_i = \left(\sum_{d=0}^{D} y_{id} - m_d x_{id} \right)^2$$

look up linear algebra proof

Logistic Regression Summary

Ⓐ$_i$

Logistic sigmoid

- not vulnerable to being **???** fit
- (other solutions)
- good for classification
- idea $f : \mathbb{R} \to (0, 1)$
- good for probability then!

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

Probability review

- Probability density
- CDF

density $= P(x_1 < x < x_2)$
CDF $= \int_{x_1}^{x_2} P(X = x)\, dx$
CDF $\approx \sum_{x=0}^{x_2} P(X = x)$
$P(X = x|Y = y) = \frac{P(X=x,Y=y)}{P(Y=y)}$
$P(Y = y) = \sum_{x \in X} P(X = x, Y = y)$

define $P(x|c_1)$, $P(x, c_2)$ $P(c_1)$, $P(c_2)$
Likelihood class

a fun experiment

$$\begin{aligned}
&= \frac{P(x|c_1)P(c_1)}{P(x|c_1)P(c_1) + P(x|c_2)P(c_2)} \\
&= \frac{P(x|c_1)P(c_1)}{P(x|c_2)P(c_2) + P(x|c_1)P(c_1)} \\
&= \frac{B}{A + B} \\
&= \frac{\frac{B}{B}}{\frac{A+B}{B}} \\
&= \frac{1}{\frac{A}{B} + 1} \\
&= \frac{1}{1 + \exp\left(\ln\left(\frac{A}{B}\right)\right)}
\end{aligned}$$

$$= \frac{1}{1 + \exp\left(-\ln\left(\frac{B}{A}\right)\right)}$$

$$= \sigma(a)$$

$$= \frac{1}{e^{-a}}$$

log ratio:

$$a = \ln \frac{P(x|c_1)P(c_1)}{P(x|c_2)P(c_2)}$$

$$a = -\ln \frac{B}{A}$$

Given x, how to decide between c_1 and c_2?

① Be racist (prejudice)

$$\frac{P(c_1)}{P(c_2)}$$ — odds of observing c_1 or c_2 in nature

② racism light:

$$P(c_1|x) = \frac{P(x|c_1)}{P(x|c_1) + P(x|c_2)}$$

— odds of observing x in c_1 situation

— odds of observing x in c_2 situation

$$P(c_1|x) = \frac{P(x|c_1)}{\sum_{c \in \{c_1, c_2\}} P(x|c)}$$

③ No prejudice:

— Latent variable explanation

forms for $P(x|c_k)\ldots$ First, count $P(x|c_k) = \frac{\sum_{c_k \in T} 1}{\sum_{c_k}}$ $T =$ true set1

impractical as $\lim_{|T| \to \infty} = \Omega\left(P(x|c_k)\right) = \infty$

— $P(x|c_k)$ — parametric, try Gaussian (scales to $|T| \to \infty$, and realistic in **???**)
$= P(x|c_k) = \mathcal{N}(x|c_k, M_k, \Sigma)$, s.t. $u_k \sim c_k$

$$P(x|c_k) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(x - M_k)^T \Sigma^{-1} (x - M_k)\right\}$$

D – dimension

Some derivation work

_ _ _ _ _ _ _ _ _ _ _ _ _ _

$$a = \ln\left(\frac{P(x|c_k)}{P(x|c_2)}\right) + \ln\left(\frac{P(c_k)}{P(c_2)}\right)$$
$$a = \ln\left(\frac{\exp\left\{-\frac{1}{2}(x - M_k)^T \Sigma^{-1} (x - M_k)\right\}}{\exp\left\{-\frac{1}{2}(x - M_2)^T \Sigma^{-1} (x - M_2)\right\}}\right) + \ln\left(\frac{P(c_k)}{P(c_2)}\right)$$

$\downarrow$ magic (quadratic terms cancel due to ??? Σ matrix)

$$\left(\Sigma^{-1}(u_1 - u_2)^T\right) x - \frac{1}{2} u_k^T \Sigma u_k + \frac{1}{2} u_k^T \Sigma^{-1} u_{k2} + \ln\left(\frac{P(c_k)}{P(c_2)}\right)$$

or $a_k = \ln\left(P(x|c_k)P(c_k)\right)$ is an independent Gaussian:

B) Multiclass logistic regression

$P(c_k|\phi) = \frac{e^{(a_k)}}{\sum_j e^{a_j}}$
discriminate
$a_k = w_k^T \phi$

Generative version
$a_k = \ln \frac{P(x|c_k)P(c_k)}{\sum_j} P(x|c_j)P(c_j)$
a lot of computation ("fuck it")

for multiclass check out

$a_k \approx w_k^T \phi$

$$
\begin{aligned}
\frac{\partial P(c_k|\phi)}{\partial a_j} &= P(c_k|\phi)\,(1 - P(c_j|\phi)) \\
\frac{\partial P(c_k|\phi)}{\partial a_j} &= P(c_k|\phi)\left(1 - \frac{e^{a_j}}{\sum_l e^{a_l}}\right) \\
&= \frac{e^{a_k}}{\sum_l e^{a_l}}\left(1 - \frac{e^{a_j}}{\sum_l e^{a_l}}\right) \\
&= \frac{e^{a_k}}{\sum_l e^{a_l}} - \frac{e^{a_k} e^{a_j}}{\sum_l \left(e^{a_l}\right)^2}
\end{aligned}
$$

$$
\begin{aligned}
&= P(c_k|\phi) - P(c_k|c_j, \phi) \\
&= P(c_k, c_j|\phi) \qquad \longleftarrow c_k \text{ without } c_j \text{ included (just } c_k\text{)}
\end{aligned}
$$

$$\frac{\partial P(c_k|\phi)}{\partial a_j} = P(c_k|\phi)\,(I_{kj} - P(c_j|\phi))$$

Feed Forward Neural Netowrks

$P(c_k|\phi) = \sigma(a_k)$

change of terms

$$P(c_k|\phi) \to y_k$$
$$c_k \sim w$$
$$y(x,w) = f\left(\sum_{j=1}^{M} w_j \phi_j(x)\right)$$
$$(\cdot) \sim \text{layer} \qquad y(x,w) = \sum_{j=1}^{M} w_j \sigma(x) \qquad y_k = \sigma(a_k)$$

$$\left(\sum_j \qquad\right) Y = \begin{bmatrix} P(c_1|\phi) \\ \vdots \\ P(c_m|\phi) \end{bmatrix}$$

$$a_j^{(x)} = w_k^T X = \sum_{i=0}^{D} w_{ji}^{(1)} + w_{j0}^{(1)} \qquad \text{(intercept term from gaussian)}$$
$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

$P(c_j|x) = \sigma(a_j)$
$y_k = \sigma(a_j)$

$$y_j(x,w) = \sigma\left(\sum_{j=1}^{M} w_j^{(1)} x_i + w_{j0}^{(1)}\right)$$

$$\downarrow$$

$$y_j(x,w) = \sigma\left(\sum_{j=1}^{M} w_j \phi_j(x) + w_{j0}\right), \quad \text{s.t. } \phi_j(x) = x_i$$
$$y_j(x,w) = \sigma\left(\sum_{j=1}^{M} w_j \phi_j^{(1)}(x) + w_{j0}\right) \quad \text{s.t. } \begin{cases} \phi_j^{(i)}(x) = \sum_{j=1}^{M} w_j \phi_j^{(i)}(x) + w_{j0} \\ \phi_j^{i=M}(x) = x_j \end{cases}$$

Training

Error function:

$$E(w) = \frac{1}{2}\sum_{n=1}^{N} \|y(x_n, w) - t_n\|^2$$

view as probability: $P(t|x, w) = \mathcal{N}(t|y(x, w), \beta^{-1})$

Do

$$P(t|X, W, \beta) = \prod_{n>1}^{N} P(t_n|x_n, w, \beta)$$

$$\downarrow$$

$$-\ln\left(P(t|x, w, \beta)\right) = \frac{\beta}{2}\sum\left\{y(x_n, w) - t_n\right\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi)$$

$$\therefore \qquad E(w) = \frac{1}{2}\sum_{n=1}^{N}\left\{y(x_n, w) - t_n\right\}^2$$

$$\boxed{\text{Probabilistic treatment} \approx \text{Sum of squared errors}}$$

1) find w_{mL} (gradient descent)

   then $\frac{1}{\beta_{mL}} = \frac{1}{N}\sum_{n=1}^{N}\left\{y(x_n, w_{mL})\right\}^2$

   for output: $\boxed{\dfrac{\partial E}{\partial a_k} = y_k - t_k \qquad (a_k \approx y_k \text{ in output units})}$

(only k varies)

$$\left(\frac{\partial E_w(\,)}{\partial a_k} = \frac{1}{2}\frac{\partial\left\{y(x_k, w) - t_k\right\}^2}{\partial a_k}\right.$$
$$= \frac{2}{2}\left\{y(x_k, w) - t_k\right\}\left(\frac{\partial y_k(x_k, w)}{\partial a_k} - 0\right)$$
$$= y(x_k, w) - t_k$$
$$\left. = y_k - t_k\right)$$

2) Back Propagation

   Ⓐ weight derivative (to do with dE)

   Ⓑ weight adjusted (application of dE)

   Childishly:

$$
\begin{aligned}
E(w) &= \sum_{n=1}^{N} E_n(w) \\
y_k &= w_{ki} x_i \\
E_n &= \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2 \qquad \text{—(input pattern } n\text{)} \\
y_{nk} &= y_k(x_n, w) \\
\frac{\partial E}{\partial w_{ji}} &= (y_{nj} - t_{nj}) x_{ni}
\end{aligned}
$$

(callback to Logistic Regression)

$a_j = \sum_i w_{ji} z_i$, $z_j = h(a_j)$, $y(\ )$

Chain rule:

$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$

or

$\frac{\partial E_n}{\partial w_{ji}} = \delta_j \frac{\partial a_j}{\partial w_{ji}}$

$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$

for outputs

$\frac{\partial E_n}{\partial w_{ji}} = (y_k - t_k) z_i$

for hiddens

$\frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$

$\delta_j = h'(a_j) \sum_k w_k; \delta_k$

Temporal Differences

(This is a type of reconfiguration)

(see Book 2 p 16 for rewrite)

Paper: On Static and Temporal Difference for MDPs (extends Reconfigurable **???**, Page 88)

want to mix a trajectory $\{l\}_{i=0}^{\infty}$ with **???** $l_{i=0}^{\infty}$, s.t. $l_i = \{s_i, a_i, s_i'\}$, $l_i \in L$

Given a trajectory, l_i^j, we want a compressed representation.

$$F_c(r_c) = \left\{ f_c : L^n \times j \to L \middle| \text{rel}\left[r_c \left(R(l_{i_1}^{i_2}) \right), r_c \left(R(l_{i_2}^{i_3}) \right) \right] = \text{rel}\left[R\left(f_c(l_{i_1}^{i_2}) \right), R\left(f_c(l_{i_2}^{i_3}) \right) \right] \right\}$$

s.t. $f_c(l_{i_1}^{i_2}) = l_{c_0}^{c_1} = l_{c_0}$

and a reward compressor

$$R_c = \{ r_c : R(L^n) \times j \to R(L) \}^{\infty}, \quad A_c = \{ f_{a_c} : L \to \{\pi_c\}^{\infty} \}$$

$$T_c(l_{c_0}, a_c \in A_c, l_{c_1})$$

$$\begin{aligned} T(s_1, \pi_c, s_2) &= \sum_{a \in \pi_c(s_1)} \pi_c(a|s_1) T(s_1, a, s_2) \\ &= \sum_{a \in \pi_c(s_1)} P(a|s_1) P(s_2|s_1, a) \\ &= \sum_{a \in \pi(s_1)} P(a, s_2|s_1) \\ &= P(s_2|s_1, \pi_c) \\ &= T(s_1, \pi_c, s_1) \end{aligned}$$

$$T(s_1, \pi_c, s_3) = \sum_{s_i \in \tilde{S}} T(s_1, \pi_c, s_i) T(s_i, \pi_c, s_2) \qquad \longleftarrow \textbf{[should this be } s_3\textbf{?]}$$

$$T(s_1, \pi_c, s_3) = T_{\pi_c}(s_1, \tilde{s}_2) \cdot T_{\pi_c}(\tilde{s}_2, s_3)$$

$$T(s_1, \pi_c, s_n) = T_{\pi_c}(s_1, \tilde{s}_2) \left(\prod_{i=2}^{n-2} T_{\pi_c}(\tilde{s}_i, \tilde{s}_{i+1}) \right) T_{\pi_c}(\tilde{s}_{n-1}, s_n)$$

$$\boxed{T(s, f_{a_c}(l), s_n) = \prod_{i=1}^{n-1} T_{\pi_c}(\tilde{s}_i, \tilde{s}_{i+1}), \qquad \tilde{s}_1 = \{s_1\}, \ \tilde{s_n} = \{s_n\}}$$

(might be intractable to learn but can guess by

(from page 85:) $\rightarrow$ link in and rewrite Both.

reconfiguration ideas:

- ☐ $g(\cdot)$ should be a learned output function (regressed)
- ☐ the states of nodes should be able to be remapped to outputs of other MDPs
- ☐ actions can be large, multi-node, set-outputs as opposed to single, discrete or continuous actions

Transitional Learning (Idea)

$\rightarrow$ The generalization of Q-Learning to a theory relates to learning transitional models

assume a process $\langle S, A, T, R \rangle$

knowing

①: $Q_t(s,a) = Q_{t-1}(s,a) + \alpha\left[R(s,a) + \gamma \max_{a^*} Q_{t-1}(s', a^*)\right]$

②: $E\left[R(s,a)\right] = \gamma \sum_{s \in S} T(s,a,s') \max_{a^*} E\left[R(s',a)\right]$

③: $\lim_{t\to\infty} Q_t(s,a) = E\left[R(s,a)\right]$

which is the general method (①) used to converge on expected reward given in ②. We believe ③.

Turns out (I think) we can do better, replacing Q-Learning with transitional learning by making threee substitutions.

④ use $P_t(s'|s,a)$ as a replacement for $T(s,a,s')$

⑤ limit recursion depth using a threshold A_{th}

⑥ model an approximation for S called $L(s)$, s.t. $L(s) \subset S$, with a basis characterized by ④

Thus we can define an approximation for ②:

$$\bar{R}_\gamma(s,a|t,\theta) = \begin{cases} \sum_{s \in L(s)} P_t(s'|s,a) \max_{a^*} \bar{R}_\gamma(s,a|t,\theta\gamma)\,, & \theta\gamma > A_{\text{th}} \\ 0\,, & \text{otherwise} \end{cases}$$

also:

$$\lim_{A_{\text{th}}\to 0} \lim_{t\to\gamma} \bar{R}_\gamma(s,a|t,\theta) = E\left[R(s,a)\right] \quad \text{if } T(s,a,s') = P_t(s'|s,a)$$

Lastly, $L(s)$ and $P_t(s'|s,a)$ can be considered. $\{s, s', w\} \in S \times S \times \mathbb{R}$ characterizes

$L(s) = s' \in \{s, s', w\}\ w > \text{th}\, B$

Financial Domain

$S = \text{Security} = \{V_S, P_S\} \qquad V_s = \{V_t\}_{t=0}^{N}\ , P_s = \{P_t\}_{t=0}^{N}$

Goal: V_{N+1}^*, P_{N+1}^*
and, a set of Securities=$\{S_i\}_{i=0}^{M}$

approaches

- predict future prices
- predict direction

Build Bayesian model

feature vector space $d = \{1, 0, -1\}$ (direction)

1) choose stocks with
   a) high beta
   b) high volume
2) find: $P\left(d \middle| \{d\}_{N-\text{frame}}^{N}\right)$? $\qquad \cdots$ assume independence
   frame $= 4 \to 81$ states
   1-week of data:
      2400 states for training
      4800 states (~ 60 in each state)
3) build $f_r\left(d \middle| \{d\}_{t=N-\text{frame}}^{N}\right) \qquad \mathcal{O}(4800)$
4) run $P(d|D) = \frac{f_r(d|D)}{\sum_{d' \neq d} f_r(d|D)}$

Testing

a) Dump f_r for all 81 states
b) Dump High/Low $P(d|D)$

Naive

Probabilistic Trading

1) $\forall S_i \in M$: calculate $f_r^i(x_t|\overbrace{x_{t_1}, x_{t-2}, x_{t-3}, x_{t-4}}^{h_{t-1}})$
   $f_r^i\,(x|h_{t-1}) = \sum_{t=\text{start\_history}}^{T} I\left\{\sum_{t'=t}^{\text{frame}+t} I\left\{d_{t'} = x_{t'}\right\} = 4\right\}$
2) $P^i(x|h_t - 1) \leftarrow \frac{f_r^i(x|h_{t-1})}{\sum_{x'} f_r^i(x'|h_{t-1})}$
   $\mathfrak{z}_t^i = I\left\{\sum_{t'=t}^{\text{fs}} I\left\{d_{t'} = x_{t'}\right\} = \text{fs}\right\}$
   $\mathfrak{z}_t^i \in \{0, 1\}$

Probabilistic Market **???**

Given a family $F = \left\{S^i, S^j, \ldots, S^k\right\}$, $|F| \in \mathbb{N}$

$f_r^F(x|h_{t-1}) = \sum_{S^i \in F} \sum_{t=\text{start\_present}} I\left\{\sum_{t'=t}\left\{d_{t'}^i = x_{t'}'\right\} = \text{frame}\right\}$
start_history $\ll$ start_present
$P^F(x|h_{t-1}) \leftarrow \frac{f_r^F(x|h_{t-1})}{\sum_{x'} f_r^F(x'|h_{t-1})}$

Probabilistic correlation **???** func:

$f_r^{i,F}\left(x^i \middle| h_{t-1}^i, h_{t-1}^j\right) = \sum_{t=\text{start\_history}}^{T} I\left\{d_t^i = x^i\right\} \cdot \mathfrak{z}_t^i \cdot \mathfrak{z}_t^j$
have access to P^i, P^F, $P^{i,j}$

7 IMPLEMENTATION

In this section implementation of the RRLN system is considered. Thus, to make this possible Section 7.1 considers heuristics used to render the theoretical system implementable. Section **??** considers experimental results.

7.1 Heuristics

7.1.1 Tractability

It is not possible to map infinite state spaces in practice, so it is advantageous to set up f_Q on a subspace.
The core goal during exploration of an MDP M_i is to learn Q-values and use Q-values as a decision-making aid.

1. Recall Section **??**, encoding: $Q(s,a) \leftarrow \sum_{s' \in S} \tilde{R}(s'|a,s)\tilde{T}(s'|a,s) + \gamma Q(s',a)$
2. $\pi_t(s,a) = \frac{e^{Q(s,a)}}{\sum_{s' \in S} 1+e^{Q(s,a)}}$, $\pi_t^*(s) = \arg\max_a \pi(s,a)$

Clearly exploring the space S each iteration is impossible, thus ??? $\mathfrak{Z}$ as a local subset. Thus using $\mathfrak{Z}_{(S)}$ it is possible to use functions f_Q and f_m to regress to $Q(S \times A)$ images.

$$\mathfrak{Z}_{(S)} \subseteq S \times A \times S$$

$\mathfrak{Z}_{(S)}$ is tractable size! and local!

$$f_m : Q_{t+1}(S \times A) \times Q_t(\mathfrak{Z}) \rightarrow Q_t(S \times A)$$

7.1.2 Learning

We also need to keep $\tilde{T}$ and $\tilde{R}_t$ updated and can employ SGD regression instances to do this. As an MDP M_i is exploring the environment the process in Section 7.1.1 will always supply the latest and most up-to-date Q-values for decision-making. Thus, the values $\tilde{T}$ and $\tilde{R}$ must be updated online to facilitate this. To do so both $\tilde{T}$ and $\tilde{R}$ can be updated with stochastic gradient descent:

$$\begin{aligned} f_T :& \tilde{T}_{t-1}(\mathfrak{Z}) \times \{s,a,s'\} \rightarrow \tilde{T}_t(\mathfrak{Z}) \\ f_R :& \tilde{R}_{t-1}(\mathfrak{Z}) \times \{R(s'|a,s)\} \rightarrow \tilde{R}_t(\mathfrak{Z}) \end{aligned}$$

④ Implementation: I use instances of stochastic gradient descent to regress $\boxed{f_T}$ and $\boxed{f_R}$

- $\tilde{T}_t(s'|a,s) \leftarrow f_T(s,a,s',T_{t-1}) = \tilde{T}_{t-1}(s'|a,s) + \alpha_T\left(\frac{f_r(s,a,s')}{f_r(s,a)} - T_{t-1}(s'|a,s)\right)$
- $\ldots\ \ldots\ \tilde{R}_t \leftarrow$ user defined / observed from sensors

where $f_r(s,a,s')$ and $f_r(s,a)$ reflect visitation frequencies.

f_Q is more difficult,

$$f_m : Q_{t+1}(s,a) \leftarrow f_Q\left(\mathfrak{Z}, \tilde{T}_t, \tilde{R}_t\right) = \sum_{s' \in \mathfrak{Z}} \tilde{T}(s'|s,a)\tilde{R}(s'|s,a) + \gamma V(s)$$

where $V(s) \approx \arg\max_a Q_{t-1}(s,a)$.

PAGE 17

On The Generalization and reuse of transitional knowledge #2

---

① Setting the state, taking a general FOMDP given usual expections (stably stochastic etc.) $m = \langle S, A, T, R \rangle$ want to find $\pi^* : \{S \times A\} \cup Q(S, A) \to A$ s. t. for some value function $V(s)$, $\pi^*(s) = \arg\max_a \sum_{s'} T\left(s'|a, s\right) R\left(s'|a, s\right) + \gamma V(s')$

Traditionally, convergence can be found directly, using stochastic gradient descent

$$Q_{t+1}(s, a) = Q_t(s, a) + \alpha \left(R(s'|s, a) - Q_t(s, a) + \gamma \arg\max_a \left(Q_t(s', a^*)\right) \right)$$

which is limited because as $Q(S, A)$ converges, it becomes difficult to adjust to changes in $R(S, A, S)$.

---

② Optimization objectives change, meaning the basis of $Q(S, A)$ is typically malleable in real-life scenarios. In this paper we present a method for separating transitional models and reward models.
We hold reward and transitional functioning separate as $\tilde{T}$ and $\tilde{R}$; and attempt to regress to true values s. t. $\tilde{T} \approx T$ and $\tilde{R} \approx R$. We then develop a Q_{map} function f_Q to **???** $Q(S, A)$ space as needed:

$$f_Q : \tilde{T}\left(S \times A \times S\right) \times \tilde{R}\left(S \times A \times S\right) \to Q_t\left(S, A\right)$$

PAGE 18

MDP: Linearization of Reward/Optimal Policy

P_a – $P_a(i,j)$ represents $T(s_i, a, s_j)$ S– all states

γ – decay factor $(0,1)$

π – policy

$V^\pi(s)$ – typical value function

$\mathbf{V}^\pi$ – vector of all values $\{V^\pi(s_1), \ldots, V^\pi(s_n)\}$

$\prec$ and $\preceq$ denote strict and non-strict vectoral inequality.

$\mathbf{R}$ – vector of reward (like $\mathbf{V}^\pi(s)$)

for optimal reward:

$$(P_{a_i} - P_a)\,(I - \gamma P_{a_i})^{-1}\mathbf{R} \succeq 0 \qquad \Leftrightarrow$$

Proof (cool as fuck):

$$a_1 \equiv \pi(s) \in \arg\max_{a \in A} \sum_{s'} P_{s_a}(s')V^\pi(s') \quad \forall s \in S$$

$$\sum_{s'} P_{s_{a_1}} \geq \sum_{s'} P_{s_a}(s')V^\pi(s') \quad \forall s \in S, a \in A$$

$$\vdots \qquad a_1 \text{ is Pareto efficient (!)}$$

$$P_{a_1}\mathbf{V}^\pi \succeq P_a V^\pi \quad \forall a \in A \backslash a_1 \qquad \text{(non-strict improvement)}$$

$$\vdots$$

$$P_{a_1}(I - \gamma P_{a_1})^{-1}\mathbf{R} \succeq P_a(I - \gamma P_{a_1})^{-1}\mathbf{R} \quad \forall a \in A \backslash a_1$$

The hard part to verify: $\mathbf{V}^\pi = (I - \gamma P_{a_1})\mathbf{R}$

PAGE 19

Transitional Learning Continued

- $\{s, s', w\}$ can be controlled to both represent the state space and accurately represent $P+(s'|s', a)$

- $A+h$, $B+h$ and γ can be controlled to speed the algorithm R_γ

- Q-learning can still be used, if calculation of $\bar{R}_\gamma$ is too "slow".

- the reward function $R(s, a, s')$ can be redefined at an instant to allow immediate re-calculation of a policy $\bar{R}_\gamma(s, a)$.

Possible experiments:

- show speed of convergence is greater, due to the "storing" of the transitional model across all actions

- show that the generalized learning allows for redefinition of the reward function.

PAGE 20

Policy

Convergence of "Bad MDP"

Question, given π_i^*, is it possible to find

$$f : \pi_j^*, \pi_k^* \to \pi_i^*$$

$$f(\pi_j(s))$$

a) suppose $f\left(\pi_j\left(s, \pi_k(s)\right)\right) = \pi_j\left(s, \pi_k(s)\right) \cup \pi_k(s)$

b) S_k,S_j – assume a subspace that is independent of effect by A, $S_j \in S_i$
   A_k, A_j – assume a subset of A_j
   T_k, T_j – assume $T(s_i, a, s_j') = 0 \quad \forall (s_i, a_j, s_j) \in S_j \times A_j \times S_j$
   R_k, R_j – assume $R(s_i, a, s_j)$
   In this case, A_j must effect $T(s_k, a_k, s_k')$ and A_k must effect $T(s_j, a_j, s_j')$
   effect how:

$$\exists a_j, a_k \sum_{s_j' \neq s_j} T\left(s_j' \middle| \begin{matrix} a_j \\ a_k \end{matrix}, s_j \right) > 0$$

$\boxed{*\text{(Derive this conclusion from } T(s_i, a_i, s_i))*}$

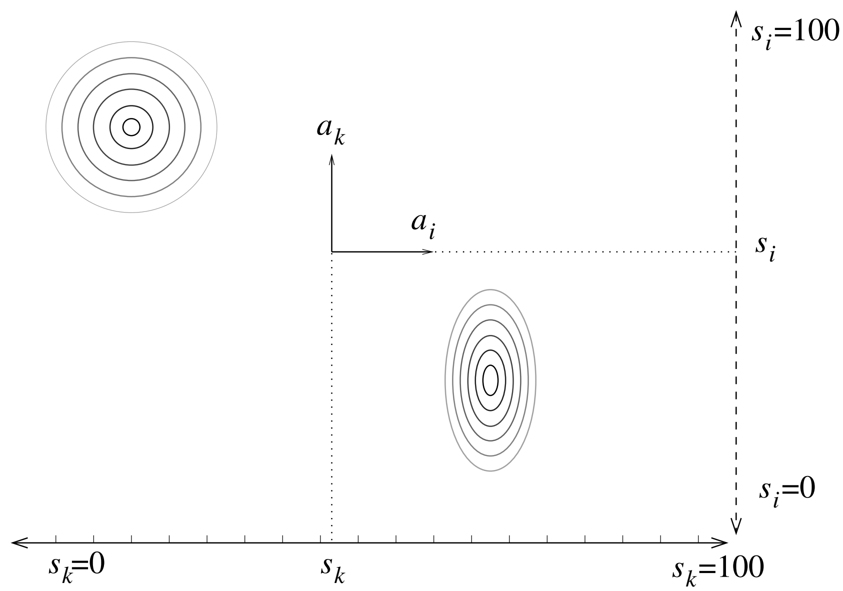
c) M_j, M_k execute concurrently, meaning at each time t $\exists (a_j, a_k) \in A_i$, chosen by $a_j \sim \pi_j(s_j, a_k) \qquad a_k \sim \pi_k(s_k)$

PAGE 21

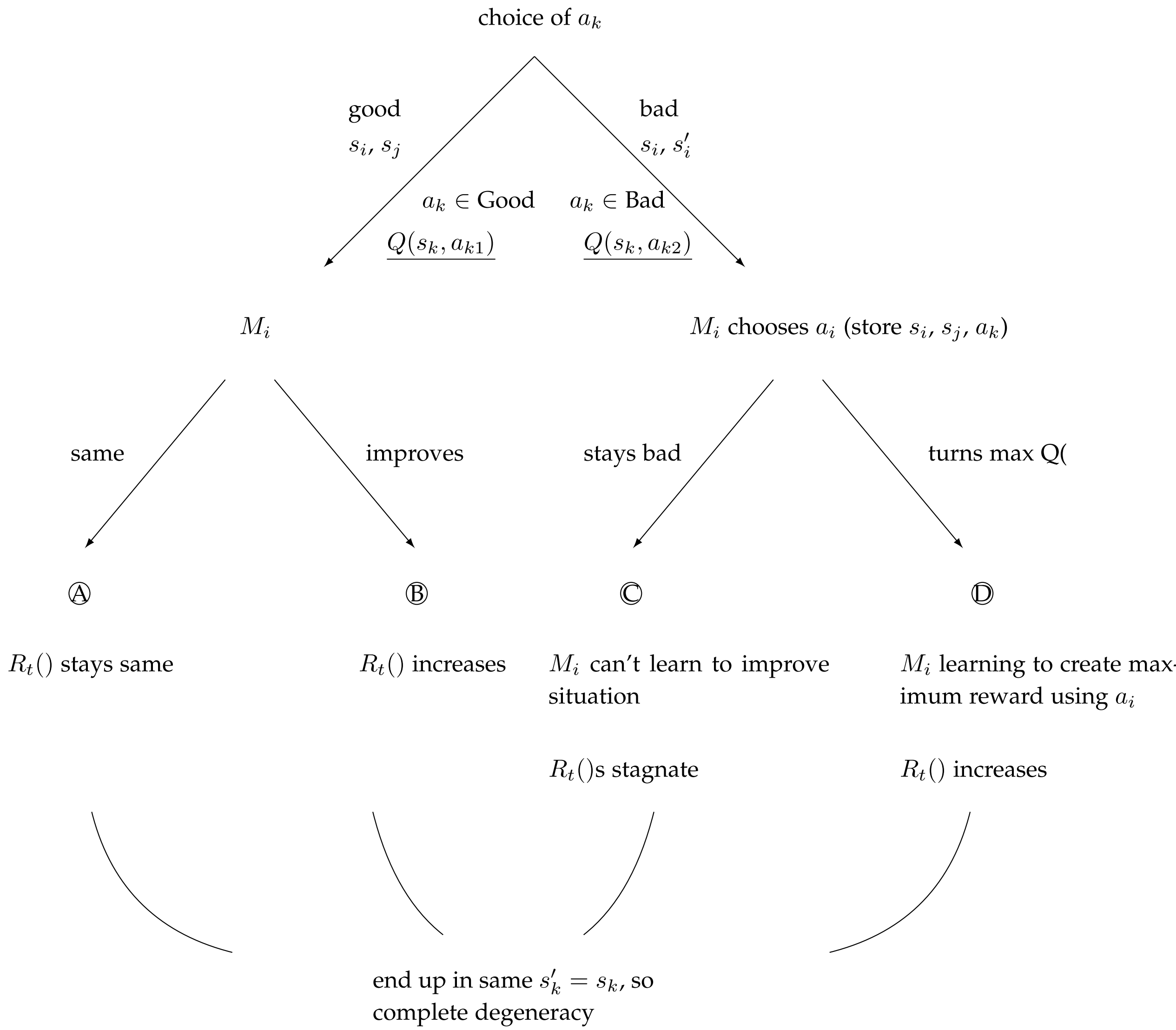
child convergence assume during concurrent learning

- R_t must be time-monotonic and convergent stochastic.

- Evaluate selection of s_i' and s_i, assuming worst case: a_k has no impact on s_k', s_k and only an impact on s_i and s_i'



consider policy learning: execution of $\pi_k(s_k)$ yielding s_k'



As $t \to \infty$, i) Ⓐ & Ⓒ will never be selected by $M_k()$

ii) if Ⓑ>Ⓓ $Q(a_k \in \text{Good}, s_k) > Q(a_k \in \text{Bad})$ and then $a_k \in$ Good will be chosen

PAGE 22

Child convergence rewrite, using time index

$m_1, \ldots, m_\alpha$

m_1: (s_i, s_k) pre-existing $s_i = 1$, $s_k = 1$

m_2: $a_k = \pi_k(s_k) = (x)$ — worst case, effects only s_i, s_j,

supposes

m_3: $a_i = \pi_i(s_i, a_k)$

$s'_i \leftarrow 2$

$s'_k \leftarrow 2$

m_4: $Q(s_k, a_k) \leftarrow$ update: $R(s_k, a_k, s'_k)$

m_5: $Q(s_i, a_k, s_i) \leftarrow$ update: $R(s_i, a_k, a_i, s'_k)$

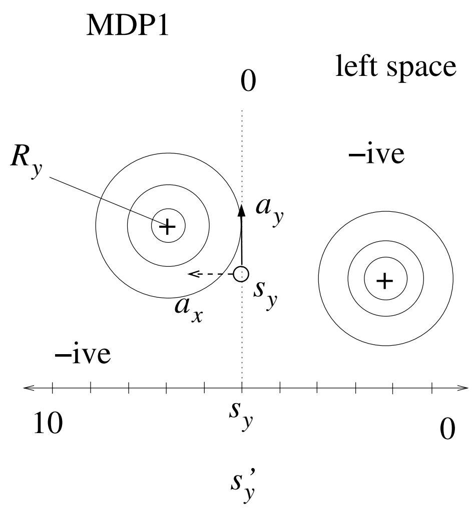
$\hookrightarrow$ update $\pi_i(s_i, a_k)$

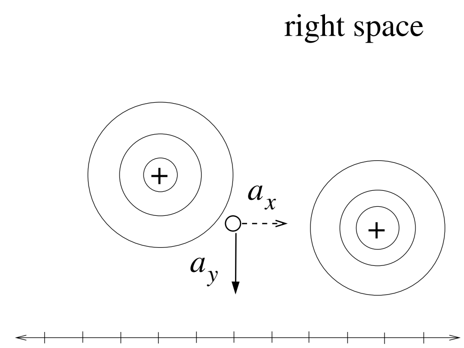
so: m_2 assumed $\pi(s_i, a_k)(m_3)$ to be convergent stochastic and monotonic in reward **???**, which is assured by updating $Q(s_i, a_k, a_i)$ at m_5

$\pi_k(s_k) \rightarrow Q(s_k, a_k)$ $\pi_i(s_i, a_k) \rightarrow Q(s_k, a_k, s_k)$

8 PAGE 23

Bad MDP Decomposition (worst possible case)





example of ''bad MDP'' which, in isolation, is useless

result – no impact on reward directly

Latent reality: move system into a position of advantage through actions

ways to understand: 1) analyze actions of subsystem

2) analyze effectiveness of subsystem

↪ Do until ave coordination

only ability: coordinate acting with subsystem, s.t., an understanding of the relationship of your action & subsystem action arises