Steps Towards Continual Learning

Satinder Singh

Professor, Computer Science & Engineering, University of Michigan
Chief Scientist, Cogitai, Inc.
Some elements of **Continual Learning**

- Learn new Skills (Options)
- Learn new Knowledge (Option-Conditional Predictions)
- Reuse / Incorporate learned Skills and Knowledge to learn more complex Skills and Knowledge [scalable, w/o catastrophic forgetting]
- Intrinsic Motivation to drive experience in the absence of (or perhaps more accurately, too long a delay in) Extrinsic Rewards
  - More experienced agents (humans) as a particularly salient target of Intrinsic Motivation (imitation, demonstration, attention, etc.)
- Increasingly competent agent over time (not just in terms of Knowledge and Skills it has but also in terms at how well it does at accumulating Extrinsic Rewards) [Learning to Learn / meta-learning]
Agent has: hand, eye, marker

Primitive Actions: 1) move hand to eye, move eye to hand, move eye to marker
move eye N, S, E, W, move eye to random object, move marker to eye,
move marker to hand. If both eye and hand are on object, operate on object
(e.g., push ball to marker, toggle light switch)

Objects: Switch controls room lights; Bell rings and moves one square if ball hits it;
Pressing blue/red block turns music on and off; Lights have to be on to see colors;
Can push blocks; Money cries out if bell and music both sound in dark room

Skills: (example)
To make monkey cry out: Move eye to switch, move hand to eye, turn lights on, move eye to blue block, move
hand to eye, turn music on, move eye to switch, move hand to eye, turn light off, move eye to bell, move marker
to eye, move eye to ball, move hand to ball, kick ball to make bell ring
Uses skills (options): turn lights on, turn music on, turn lights off, ring bell

Singh, Barto & Chentanez
Options (Precup, Sutton, & Singh, 1997)

A generalization of actions to include temporally-extended courses of action

An option is a triple $o = <I, \pi, \beta>$

- $I$: initiation set: the set of states in which $o$ may be started
- $\pi$: is the policy followed during $o$
- $\beta$: termination conditions: gives the probability of terminating in each state

Example: robot docking

- $I$: all states in which charger is in sight
- $\pi$: pre-defined controller
- $\beta$: terminate when docked or charger not visible
Coverage of Continual Learning elements?

**Intrinsic reward** proportional to *error in prediction of an (salient) event according to the option model for that event* (“surprise”); Motivated in part by the novelty responses of dopamine neurons; Behavior determined by this intrinsic reward.

- *(Cheat)* Built in salient stimuli: changes in light intensity, changes in sound intensity

**Incremental Creation of Skills/Options:** Upon first occurrence of salient event create an option for that event and add it to skill-KB; initialize its policy, termination conditions, etc.

**Incremental Creation of Predictions/Knowledge:** Upon initiating an Option, initialize and start building an Option-Model

**Updating Skills/Knowledge:** All options and option-models are updated all the time using intra-option learning *(learning multiple skills and knowledge in parallel)*

**Reuse of Skills/Knowledge to Learn Increasingly Complex Skills/Knowledge:** Use model-based RL (with previously learned options as actions) to learn new skills/knowledge.
Hierarchy of Reusable Skills

- **Hardwired Primitive Options**
  - Saccade to random object
  - Marker to eye

- **Activate Toy**
  - Turn Music On
  - Turn Light On

- **Turn Music Off**
  - Turn Light Off
  - Ring Bell
Do the Intrinsic Motivations Help?

Performance of Learned Options

Effect of Intrinsically Motivated Learning

Average Number of Actions to Salient Event

Number of Actions

Number of extrinsic rewards

Sound On
Light On
Music On
Toy Monkey On

Number of steps between extrinsic rewards

Extrinsic Reward Only
Intrinsic and Extrinsic Rewards
Discussion

1. Learned new Skills/Options
2. Learned new knowledge in the form of predictions for the new Skills (option-models)
3. Reused learned Skills to learn more complex Skills (and associated Knowledge)
4. Agent got more competent over time at Extrinsic Reward

Caveats:
(Extremely) Contrived domain
Intrinsic Motivations were about hard-wired salient events; very limited form of intrinsic reward.
All Lookup Tables (and so scaling and catastrophic forgetting/interference not present)

Next: On Deriving Intrinsic Motivations
On the Optimal Reward Problem*
(Where do Rewards Come From?)

Satinder Singh

*with Nuttapong Chentanez, Andrew Barto, Jonathan Sorg, Xiaoxiao Guo & Richard Lewis
Autonomous Agent Problem

- Env. State Space $S$
- Agent Action Space $A$
- Rewards $R: S \rightarrow \text{scalars}$
- Policy: $S \rightarrow A$

Agent’s purpose is to act so as to maximize expected discounted sum of rewards over a time horizon (the agent may or may not have a model to begin with).
Preferences-Parameters Confound

• (Most often the) starting point is an agent-designer that has an *objective reward function* that specifies preferences over agent behavior (it is often way too sparse and delayed)

• What should the agent’s reward function be?

• A single reward function confounds two roles (from the designers point of view) simultaneously in RL agents
  1. *(Preferences)* It expresses the agent-designer’s preferences over behaviors
  2. *(Parameters)* Through the reward hypothesis it expresses the RL agent’s goals/purposes and becomes parameters of actual agent behavior

These roles seem distinct; should they be confounded?
Revised Autonomous Agent

Agent reward is *internal* to the agent
Parameters to be designed by agent-designer
Approaches to designing reward

• Inverse Reinforcement Learning (Ng et.al.)
  • Designer/operator demonstrates optimal behavior
  • Clever algorithms for automatically determining set of reward function such that observed behavior is optimal (e.g., Bayesian IRL; Ramachandran & Amir)
  • Ideal: Set agent reward = objective reward (i.e., preserve the preferences parameters confound)

• Reward Shaping
  • (Ng et.al.) agent reward = objective reward + potential-based reward (breaks PP confound)
  • Objective: To achieve agent with objective reward’s asymptotic behavior faster! [Also Bayesian Reward Shaping by Ramachandran et.al.]

• Preference Elicitation (Ideal: preserves PP confound)
• Mechanism Design (in Economics)
• Other Heuristic Approaches
Optimal Reward Problem

- There are two reward functions
  1) Agent-designer’s: objective reward $R_O$ (given)
  2) Agent’s: reward $R_I$

Agent $G(R_I;\Theta)$ in Environment $Env$ produces (random) interaction $h \sim <Env,G(R_I;\Theta)>$

Utility of interaction $h$ to agent is $U_I(h) = \Sigma_t R_I(h_t)$
Utility to agent designer is $U_O(h) = \Sigma_t R_O(h_t)$

Optimal Reward $R^*_I =$

$$\arg \max_{R_i \in \{R_i\}} \mathbb{E}_{\mathbb{E}_{h \sim <Env,G(R_i;\Theta)>}} \left\{ U_O(h) \right\}$$

Nested Optimizations; Outer reward opt.; Inner Policy opt.
Illustration: Fish-or-Bait

E: Fixed location for fish and bait
A: movement actions, eat, carry
A: observes location & food, bait when at those locations & hunger-level & carrying-status
Bait can be carried or eaten
Fish can be eaten only if bait is carried on agent
Eat fish -> not-hungry for 1 step
Eat bait -> med-hungry for 1 step else hungry
Agent is a lookuptable Q-learner

Objective utility: $U_0(h)$ increment of 1.0 for each fish & 0.04 for each bait eaten (but to reduce sensitivity of precise numbers chosen we will search over additive constants)
Reward Space

Reward features: hunger-level (3 values) (thus generalization across location is built in!)

Multiple experiments: for varying lifetimes/horizons

Reward features:
- hunger-level (3 values)
  (thus generalization across location is built in!)

Multiple experiments: for varying lifetimes/horizons

Life length at which agent has enough time to learn to eat fish with internal reward

Life length at which agent has enough time to learn to eat fish with designer’s reward

> by 3.0
(PP Confound Matters?) Mitigation

Increasing Agent-Designer Utility

Unbounded agent with confounded reward

Bounded agent with optimal reward

Bounded agent with confounded reward
Policy Gradient for Reward Design (PGRD)

(Sorg, Singh, Lewis; NIPS 2010)

Insight: In planning agents, the reward function parameterizes the agent’s policy

![Diagram]

PGRD optimizes the reward function via a standard Policy Gradient (PG) approach (OLPOMDP [2])
PGRD...

- Optimizes Reward for Planning Agents for Full depth-D planning as well as for the much more practical UCT
- Computes D-step action values $Q^D(s,a)$
- Selects actions using Boltzmann distribution parameterized by action-values $Q^D$; policy denoted $\mu$
- Agent reward is parameterized by $R(:,\Theta)$
- PGRD approximates gradient in 2 parts:

\[ \nabla_\theta \mathbb{E}[R_O(trajectory)|Agent(R(\cdot; \theta))] = \nabla_\mu \mathbb{E}[R_O(trajectory)|\mu] \times \nabla_\theta \mu(s, a; \theta) \]

- Gradient of performance w.r.t. the policy
- Approximated by OLPOMDP

\[ \nabla_\theta Q^D(s, a) = \nabla_\theta R(s, a; \theta) + \sum_{s', a'} T(s'|s, a) \pi(a|s) \nabla_\theta Q^{D-1}(s', a') \]
Deep Learning for Reward Design to Improve UCT in ATARI (IJCAI 2016)
Forward View: From Rewards to Utility

- Monte Carlo average of root node  ➔  Execution policy of UCT

\[ Q(s\downarrow 0 \uparrow N, b) = \sum_{i' = 0}^{N - 1} 1 \sum_{h' = 0}^{H - 1} \gamma^{h'} \left[ R(s\downarrow h' \uparrow i', a\downarrow h' \uparrow i') + \text{CNN}(s\downarrow h' \uparrow i', a\downarrow h' \uparrow i'; \theta) \right] \]

- UCT’s utility:

\[ \theta^{*} = \text{argmax}_\theta \mathbb{E}\{\sum_{t=0}^{T - 1} R(s\downarrow t, a\downarrow t) | \theta\} \]

Backward View: From Utilities to CNN gradients

• Monte Carlo average of root node

\[
Q(s_{0:N}, b) = \sum_{i' = 0}^{N-1} \frac{1}{n(s_{0:N}, b, 0)} \sum_{h' = 0}^{H-1} \gamma^{h'} [R(s_{h'}, i', a_{h'}, i') + \text{CNN}(s_{h'}, a_{h'}, \theta)]
\]

• Real execution policy of UCT in learning:

\[
\theta^{*} = \arg\max_{\theta} \mathbb{E}\{\sum_{t=0}^{T-1} R(s_t, a_t) | \theta}\}
\]

• UCT’s utility:

Gradient calculation and variance reduction details can be found in the paper.
Main Results: improving UCT

- 25 ATARI games
- 20 games have ratio larger than 1
- Not an apples-to-apples comparison
  - ignores the computational overhead for reward bonus
- An apples-to-apples comparison
  - comparison with UCT with same time cost per decision (i.e. deeper or wider UCT)
- 15 games have ratio larger than 1
Repeated Inverse Reinforcement Learning (for Lifelong Learning agents)

Satinder Singh*
Computer Science and Engineering
University of Michigan

May 2017
*with Kareem Amin & Nan Jiang
Inverse Reinforcement Learning

[Ng&Russell’00] [Abbeel&Ng’04]

• Input
  - Environment dynamics
e.g., an MDP without a reward function
  - Optimal behavior
e.g., the full policy or trajectories

• Output: the inferred reward function
Unidentifiability of Inverse RL

- Bad news: problem fundamentally ill-posed
Unidentifiability of Inverse RL

[Ng&Russell’00] The set of possible reward vectors is:

\[ \{ v : \forall a, (P^{\pi^*} - P^a)(I - \gamma P^{\pi^*})^{-1} v \geq 0 \} \]

- Bad news: problem fundamentally ill-posed
- Good news (?): may still mimic a good policy for \textit{this task} even if reward is not identified

And yet...
An example scenario:

- **Intent:** background reward function $\theta^* : S \rightarrow [-1, 1]$
  - no harm to humans, no breaking of laws, cost considerations, social norms, general preferences, ...

- Multiple tasks: $\{(E_t, R_t)\}$
  - $E_t = \langle S, A, P_t, \gamma, \mu_t \rangle$ is the *task environment*
  - $R_t$ is the *task-specific reward*

- Assumption: human is optimal in $\langle S, A, P_t, R_t + \theta^*, \gamma \rangle$

Can we learn $\theta^*$ from optimal demonstrations on a few tasks **OR** generalize to new ones?
Looking more carefully at unidentifiability

There are two types of unidentifiability in IRL.

(1) Representational Unidentifiability
    Should be ignored.

(2) Experimental Unidentifiability
    Can be dealt with.
Representational Unidentifiability

Behavioral Equivalence

We say two reward functions $R$ and $R'$ are *behaviorally equivalent* if they induce the same set of optimal policies in *any possible environment* $E$.

For any $E$, the MDP $(E, R)$ has the same set of optimal policies as $(E, R')$.

- Behavioral equivalence induces equivalence classes $[R]$ over rewards.
- For each $[R]$, fix a canonical element of $[R]$.

Goal of Identification is to find canonical element of $[\theta_*]$
“Experimenter” chooses tasks

Formal protocol

- The experimenter chooses \( \{(E_t, R_t)\} \)
- Human subject reveals \( \pi_t^* \) (optimal for \( R_t + \theta_* \) in \( E_t \))

Theorem: If any task may be chosen, there is an algorithm that outputs \( \theta \) s.t. \( \|\theta - \theta_*\|_\infty \leq \varepsilon \) after \( O(\log(1/\varepsilon)) \) tasks.
“Experimenter” chooses tasks

Theorem: If any task may be chosen, there is an algorithm that outputs $\theta$ s.t. $\|\theta - \theta^*\|_{\infty} \leq \varepsilon$ after $O(\log(1/\varepsilon))$ tasks.

Uncertainty in $\theta^*$

$R_i$

$\theta^*$ (unknown)
“Experimenter” chooses tasks

Theorem: If any task may be chosen, there is an algorithm that outputs $\theta$ s.t. $\|\theta - \theta^*\|_\infty \leq \varepsilon$ after $O(\log(1/\varepsilon))$ tasks.
Issue with the Omnipotent setting

• Motivation was the difficulty for a human to specify the reward function

• But in the experiment, we ask: “would you want something if it costs you $X?”

• Can we make weaker assumptions on the tasks?
Nature chooses tasks

Given a sequence of arbitrary tasks \( \{(E_t, R_t)\} \) …

1. Agent proposes a policy \( \pi_t \).
2. If near-optimal, great!
3. If not, a mistake is counted, and human demonstrates \( \pi_t^* \) (optimal for \( R_t + \theta^* \) in \( E_t \)).

Algorithm design: how to behave (i.e., choose \( \pi_t \))?

Analysis: upper bound on the number of mistakes?
Value and loss of a policy

Given task \((E, R)\) where \(E = \langle S, A, P, \gamma, \mu \rangle\), the (normalized) value of a policy \(\pi\) is defined as:

\[
(1 - \gamma) \mathbb{E} \left[ \sum_{\tau=1}^{\infty} \gamma^{\tau-1} (R(s_\tau) + \theta_\ast(s_\tau)) \mid s_1 \sim \mu_1, \pi, P \right]
\]

which is equal to \(\langle R + \theta_\ast, \eta_{\mu,P}^{\pi} \rangle\) where

\[
\eta_{\mu,P}^{\pi} = (1 - \gamma) (\mu^T (I - \gamma P^{\pi})^{-1})^T
\]

discounted occupancy vector (\(\|\eta_{\mu,P}^{\pi}\|_1 = 1\))

Define

\[
\text{loss} = \langle R + \theta_\ast, \eta_{\mu,P}^{\pi_\ast} - \eta_{\mu,P}^{\pi} \rangle
\]
Reformulation of protocol

Every environment $E$ induces a set of occupancy vectors $\{x^{(1)}, x^{(2)}, \ldots, x^{(K)}\}$ in $\mathbb{R}^d$ (“arms”).

1. Agent proposes $x$. Let $x^*$ be the optimal choice.
2. If $\langle \theta^* + R, x \rangle \geq \langle \theta^* + R, x^* \rangle - \varepsilon$, great!
3. If not, a mistake is counted, and $x^*$ is revealed.

Formally, we use transformation to Linear Bandits
Algorithm outline

Let $\theta$ be some guess of $\theta_*$ and behave accordingly:

$$\langle \theta, x \rangle \geq \langle \theta, x^* \rangle \quad (1)$$

If a mistake is made:

$$\langle \theta_*, x \rangle < \langle \theta_*, x^* \rangle \quad (2)$$

(2) - (1):

$$\langle \theta_* - \theta, x^* - x \rangle > 0$$

For simplicity, assume for now that $R = 0$

How to choose $\theta$?
The ellipsoid algorithm

**Theorem**: the number of total mistakes is $O(d^2 \log(d/\varepsilon))$. 

$x^*$ does not have to be optimal; it just has to be better than $x$.
Experimenter chooses tasks
choose \( \{(E_t, R_t)\} \) to identify \( \theta^* \)
\( \log(1/\varepsilon) \) demo’s

Nature chooses tasks
choose \( \{\pi_t\} \) to minimize loss
\( O(d^2 \log(d/\varepsilon)) \) demo’s

\( \Omega(d \log(1/\varepsilon)) \) lower bound

gap?
Zero-Shot Task Generalization by Learning to Compose Sub-Tasks

Satinder Singh

Junhyuk Oh, Honglak Lee, Pushmeet Kohli
Rapid generalization is key to Continual Learning

• Humans can easily infer the goal of unseen tasks from similar tasks even without additional learning.
  • e.g.,) Pick up A, Throw B \rightarrow Throw A ?

• When the task is composed of a sequence of sub-tasks, humans can also easily generalize to unseen compositions of sub-tasks.
  • e.g.,) Pick up A and Throw B \rightarrow Throw B and Pick up A ?

• Imagine a household robot that is required to execute a list of jobs. It is infeasible to teach the robot to do every possible combination of jobs.
Problem: Instruction Execution

• Given
  • Randomly generated grid-world
  • A list of instructions as natural language
• Goal: execute instructions
• Some instructions require repetition of the same sub-task
  • e.g.,) Pick up “all” eggs
• Random event
  • A monster randomly appears.

Instruction
Visit cow
Pick up diamond
Hit all rocks
Pick up all eggs
Challenges

• Solving unseen sub-tasks itself is a hard problem.
• Deciding when to move on to the next instruction.
  • The agent is not given which instruction to execute.
  • Should detect when the current instruction is finished.
  • Should keep track of which instruction to solve.
• Dealing with long-term instructions and random events.
• Dealing with unbounded number of sub-tasks.
• Delayed reward
Overview

- **Multi-task controller**: 1) execute primitive actions given a sub-goal and 2) predict whether the current sub-task is finished or not.
Overview

- **Multi-task controller**: 1) execute primitive actions given a sub-goal and 2) predict whether the current sub-task is finished or not.

- **Meta controller**: set sub-goals given a description of a goal.
Goal Decomposition

• A sub-goal is decomposed into several arguments.
Multi-task Controller Architecture

- **Given**
  - Observation
  - Sub-goal arguments

- **Do**
  - Determine a primitive action
  - Predict whether the current state is terminal or not
Analogy Making Regularization

- Desirable property

\[ \varphi(g_1) - \varphi(g_2) \approx \varphi(g_3) - \varphi(g_4) \]
\[ \|\varphi(g_1) - \varphi(g_2)\| \geq \tau_{diff} \gg 0 \]

if \( g_1 - g_2 = g_3 - g_4 \)
if \( g_1 \neq g_2 \)
Analogy Making Regularization

- Objective function (contrastive loss)

\[
\mathcal{L}_{sim} = \mathbb{E}_{(g_1, g_2, g_3, g_4) \sim g_{sim}} \left[ \| \varphi(g_1) - \varphi(g_2) - (g_3) + \varphi(g_4) \|^2 \right]
\]

\[
\mathcal{L}_{diff} = \mathbb{E}_{(g_1, g_2) \sim g_{diff}} \left[ \max(0, \tau_{diff} - \| \varphi(g_1) - \varphi(g_2) \|^2) \right]
\]

\[
\mathcal{L}_{AM} = \mathcal{L}_{sim} + \rho \mathcal{L}_{diff}
\]
Multi-task Controller: Training

• **Policy Distillation** followed by **Actor-Critic** fine-tuning
  - Policy Distillation: Train a separate policy for each sub-task and use them as teachers to provide actions (labels) for the multi-task controller (student) in supervised learning setting

• **Final objective**
  - RL objective + Analogy making + Termination prediction objective

Policy Distillation or Actor-Critic

Binary classification
Meta Controller

Meta Controller

Multi-task Controller

Terminal

Sub-goal

Arg 1

Arg n

Goal

Observation

Action
Meta Controller Architecture

• **Given**
  - Observation
  - Current sub-goal
  - Current instruction
  - Current sub-task termination

• **Do**
  - Determine which instruction to execute
  - Set a sub-goal

![Diagram of Meta Controller Architecture](diagram.png)
Meta Controller: Learning Temporal Abstraction

• Motivation
  • The meta controller operates at a high-level (sub-goal).
  • It is desirable for the meta controller to operate in larger time-scale.

• Goal: Update the sub-goal and the memory pointer only when it is needed

• Method
  • Decide whether to update the sub-goal or not (binary decision)
  • If yes, update the memory pointer and update the sub-goal
  • If no, continue the previous sub-goal
Meta Controller: Learning Temporal Abstraction

\[
\text{Update} \quad \begin{array}{c}
\text{No update} \\
\end{array} \\
\begin{array}{c}
+1 \\
0 \\
-1 \\
\end{array}
\]

\[
\begin{array}{c}
\text{Visit A} \\
\text{Pick up B} \\
\text{Hit C} \\
\text{Pick up D} \\
\end{array}
\]

Do forward propagation only when update == true

\[
\begin{array}{c}
\text{Current sub-goal} \\
\varphi(g_{t-1}) \\
\end{array} \\
\begin{array}{c}
\text{Sub-task} \\
\text{termination} \\
\end{array}
\]

\[
\begin{array}{c}
x_{t-k:t} \\
\text{Conv} \\
h_t \\
g_t^{(1)} \\
g_t^{(2)}
\end{array}
\]
Meta Controller: Learning Temporal Abstraction

Update

No update

Copy the previous sub-goal

$\varphi(g_{t-1})$ 

Sub-task termination

$g_t^{(1)} \leftarrow g_{t-1}^{(1)}$

$g_t^{(2)} \leftarrow g_{t-1}^{(2)}$
Does it Work?
Value Prediction Networks*

Junhyuk Oh, Satinder Singh, Honglak Lee

*Under Review (on arXiv in Late July, 2017)
Motivation

• Observation Prediction (Dynamics) Models are difficult to build in high-dimensional domains.

• We can make lots of prediction at different temporal scales

• So, how do we plan without predicting observations?

VPNrs are heavily inspired by Silver et.al’s Predictron
Predictron was limited to Policy Evaluation
VPNrs extend to Learning Optimal Control
VPN: Architecture

Encoding $f_{\theta}^{enc}: x \mapsto s$
Outcome $f_{\theta}^{out}: s, o \mapsto r, \gamma$
Value $f_{\theta}^{value}: s \mapsto V_{\theta}(s)$
Transition $f_{\theta}^{trans}: s, o \mapsto s'$
Planning in VPNs
Learning in VPNs

Perform d-step planning to get
\[
\max_o Q^d_\theta(s_{n+1}, o)
\]
Collect Domain: Results 1

Domain

DQN Traj.

VPN Traj.
Collect Domain: Results 2

VPN *Plan (20 steps)*

VPN *Plan (12 steps)*
Collect Domain: Comparisons

Average reward vs Epoch for Greedy, Shortest, DQN, OPN(1), OPN(2), OPN(3), OPN(5), VPN(1), VPN(2), VPN(3), VPN(5)
VPN: Results on ATARI Games
Questions?