Integrating BDI and Reinforcement Learning

Michael Bosello   Alessandro Ricci   Giovanni Pau

Alma Mater Studiorum – Università di Bologna
Department of Computer Science and Engineering, Cesena Campus

michael.bosello@unibo.it

- From Programming Agents to *Educating* Agents
  A Jason-based Framework for Integrating Learning in the Development of Cognitive Agents
- Integrating BDI and Reinforcement Learning: the Case Study of Autonomous Driving

Outline

Next in line...

Vision

Machine learning

- Machine learning will be increasingly
  - an important feature for software products [Jones, 2014]
  - side by side with developers [Meijer, 2018] [Karpathy, 2017]
- Intersection of learning and software engineering still needs to be explored [Arpteg et al., 2018] [Khomh et al., 2018]
  - we pursue engineering features like modularity, reusability, testability

Purpose

- We want to systematically exploit ML in (agent-oriented) programming activities
- Promising direction toward *software 2.0 era*

Basic idea

- The developer writes some plans and let the agent learns others
- Use them in a seamless way
- As a feature of the agent platform

Next in line...

Agents and learning integration

- A main research topic in agents and MAS literature since their roots [Weiß, 1996]

Adaptivity

- A lot of works focus on the adaptivity problem
  - in particular on improving the plans selection
- Some references: [Guerra-Hernández et al., 2004] [Singh and Hindriks, 2013] [Singh et al., 2011] [Norling, 2004] [Airiau et al., 2009]

Other approaches

- Using Jason to implement RL methods [Badica et al., 2015] [Badica et al., 2017]
  - to face the RL problem with a more appropriate paradigm
- Once a policy is learned, a BDI agent is generated from it [Feliú, 2013]

Instances of BDI & learning

BDI-FALCON architecture

- [Tan et al., 2011] is a great example of BDI-learning integration
- TD-FALCON is a neural network based reinforcement learner
- Goals are represented with a target vector and an attainment function that defines the degree of achievement – used for reward
- Plans have confidence proportional to the probability of success
- The FALCON module updates the confidences according to the outcome of a plan execution
- If a plan is not available, the FALCON module decides the actions to perform and, when a successful sequence is found, a new plan is crafted

- Also [Karim et al., 2006] proposes a hybrid BDI-FALCON architecture

Differences between existing works and our proposal

- We focus on the representation of the state/action/reward and the management of the learning modules in a cognitive architecture

$\Rightarrow$ How to seamless integrate RL in the reasoning cycle

- The developer decides when using learning and for which tasks
- We seek to manage the boundary between adaptivity and controllability

Next in line...

Main idea

Hard Plans and Soft Plans

- The development process is extended with a learning phase – *education*
- The developer can:
  - write some plans — ***Hard Plans***
  - let the agent itself learn other plans — ***Soft plans***
- At runtime they are treated in a uniform way
- For soft plans, one must set up the learning phase

Learning module

- With soft plans we obtain a notion of *learning module*
- We can *replace* it, *reuse* it, *test* it

Concepts representation in BDI I

RL	Proposed BDI+ construct	Representation in BDI
Observations	Belief about Learning	Belief subset
Actions	Actions	Relevant Plans
Rewards	Motivational Rule	Belief Rule
Episode	Terminal Rule	Belief Rule
Policy	Learning Module	Plan
Behavior	Behavior	Intention

Model

- It represents the general concepts of RL
- It abstracts from the specific algorithm and problem

Concepts representation in BDI II

RL	Proposed BDI+ construct	Representation in BDI
Observations Actions	Belief about Learning Actions	Belief subset Relevant Plans

Belief about Learning

- Belief useful to the learning process

Actions

- Subset of the plan library – plans can represent simple actions and compound actions
- Different levels of planning granularity
- It is called *option framework* in RL
- Context of plans used to define different action sets for different states

Concepts representation in BDI III

RL	Proposed BDI+ construct	Representation in BDI
Rewards	Motivational Rule	Belief Rule
Episode	Terminal Rule	Belief Rule
Policy	Learning Module	Plan
Behavior	Behavior	Intention

Motivational Rule

- Reflects the agent desires
- Generators of internal stimuli in the agent like a reward signal in neuroscience

Terminal Rule

- When one of these rules evaluates to true, the episode ends

RL Reasoner

- Elaborates the information and execute the learning algorithm
- Defines the behavior by suggesting the action to perform

Graphic representation of BDI-RL

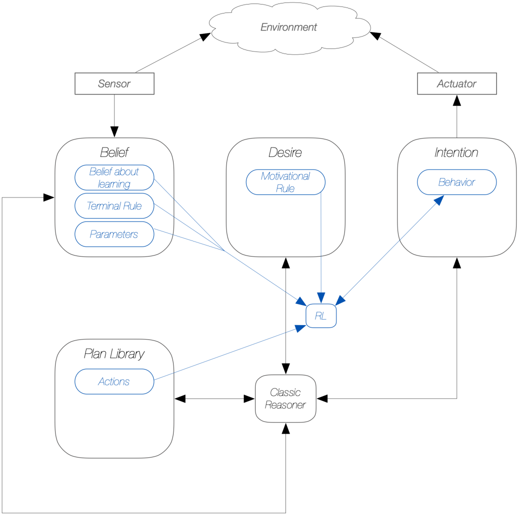


Figure: A graphic representation of the BDI model with the addition of our constructs

Extended BDI practical reasoning

```
1.  B ← B'; /* B' are initial beliefs */
2.  I ← I';   /* I' are initial intentions */
3.  RL Algorithm parameters: step size α ∈ (0, 1], small ε > 0
4.  Initialize Q(s, a), for all s ∈ S+, a ∈ A(s), arbitrarily except that Q(terminal, ·) = 0
5.
6.  while true do
7.     get next percept ρ via sensors;
8.        B ← brf(B, ρ);
9.        D ← options(B, I);
10.       I ← filter(B, D, I);
11.       π ← plan(B, I, Ac); /* Ac is the set of actions */
12.       while ( (π is learning plan and S is not terminal) or
13.              not (π is learning plan and empty(π)) ) and
14.              not (succeeded(I, B) or impossible(I, B)) do
15.          if π is learning plan then
16.             O ← related(B, I)
17.             S' ← state(O)
18.             R ← motivationalRule(B)
19.             Choose A' from A(S') using policy derived from Q //(e.g. ε-greedy)
20.             If S not null and A not null then
21.                Q(S, A) ← Q(S, A) + α[R + γ*Q(S', A') - Q(S, A)]
22.             end-if
23.             S ← S'
24.             A ← A'
25.             execute(A)
26.          else
27.             α ← first element of π;
28.             execute(α);
29.             π ← tail of π;
30.          end-if
31.          observe environment to get next percept ρ;
32.          B ← brf (B, ρ);
33.          if reconsider(I, B) then
34.             D ← options(B, I);
35.             I ← filter(B, D, I);
36.          end-if
37.          if not sound(π, I, B) then
38.             π ← plan(B, I, Ac)
39.          end-if
40.       end-while
41. end-while
```

Practical reasoning extended with RL

- The figure shows the pseudo code of a classic BDI agent reasoning cycle
- Extended with learning capabilities, specifically with the SARSA algorithm (adapted for the context)
- Additions are in red
- The function 'plan' in (11) is extended to include also soft plans

Next in line...

Framework Proof of Concept

Jason implementation

- We developed a PoC of the framework in Jason
- The state and action spaces dimensions are critical factor for an effective learning

⇒ The framework must make it possible for reduce the state/action spaces

- Notion of soft-plan (with independent contexts)
- Definition of states/actions/rewards/termination for each soft-plan

Reference example: gridworld

- The agent can move in four directions: right, left, up, down
- The agent must reach a target block doing the minimum number of steps

Implementation of the framework available on GitHub

- `https://github.com/MichaelBosello/jacamo-rl`

Constructs I

Soft plan goal

- A ground term identifies a soft plan goal

⇒ Enables learning of multiple tasks

Syntax `g`

Example `reach_end`

Belief about learning

- Declares the relevant beliefs for a task

⇒ Reduce the state space

Syntax `rl_observe(g, [` o_1`,` o_2`, ...` o_n`]).`

Example `rl_observe(reach_end, [ position ]).`

Constructs II

Motivational rule

- *R* is the reward and must be a number, possibly a variable set in the rule's body
- The final reward is the sum of all the rewards provided by the currently true motivational rules

Syntax
```
rl_reward(g, R) :- <reward conditions>.
```

Example
```
rl_reward(reach_end, 10) :- finish_line.
rl_reward(reach_end, -1) :- not finish_line.
```

Terminal rule

- When one of these rules evaluates to true, the task *g* ends

Syntax
```
rl_terminal(g) :- <episode ends conditions>.
```

Example
```
rl_terminal(reach_end) :- finish_line.
```

Constructs III

Action

- The action label declares the plans useful for a task

⇒ Reduce the action space

- One can use variables in action declaration. Variable type and range must be indicated in the label

⇒ Allows to specify *range* of actions (even *continuous* one)

Syntax `@action[rl_goal(g1, ..., gn), rl_param(p1, ..., pm)]`

Example
```
@action[rl_goal(reach_end),
        rl_param( direction(set(right, left, up, down)) )]
+!move(Direction) <- move(Direction).
```

Constructs IV

Learning parameters

- Also the learning parameters are entered as beliefs

⇒ Allows the complete control of the learning process by the agent/developer

⇒ The agent can manage the exploration/exploitation phases

Syntax

```
rl_parameter(name, value).
```

Example

```
rl_parameter(alpha, 0.26).
rl_parameter(gamma, 0.9).
rl_parameter(policy, egreedy).
```

Constructs V

Soft plan learning and execution

- The internal action *rl.execute* run the soft plan to achieve *g*
- In the meantime, the reasoner updates (learns) the policy

⇒ The action performs one learning run

One episode for an episodic task
Goes on continuously for a continuous task

- ▷ (one can set a limit e.g. performance, time, number of actions)

- One can use a belief parameter to choose between learn-and-act or act only
- The internal action is implemented in Java, this allows to reuse existing RL libraries

Syntax `rl.execute(g)`

Constructs VI

Soft plan evaluation

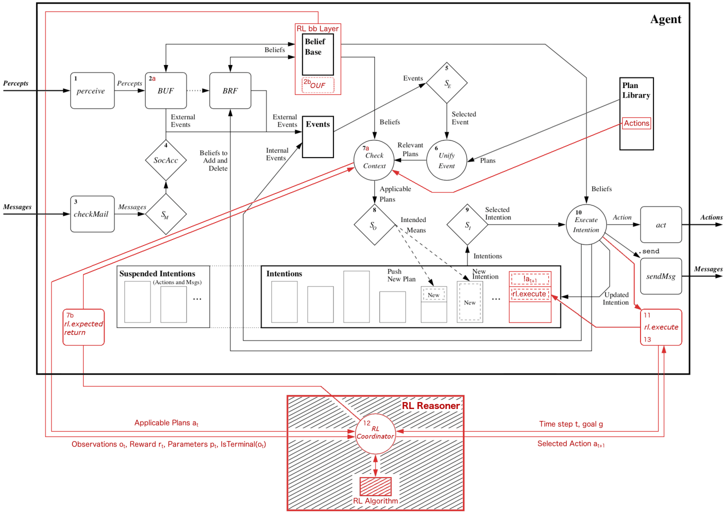
- The internal action *rl.expectedreturn* gets the estimate of future rewards *R* for the goal *g* on the basis of the current state and learned policy, i.e. the *expected return*
- Could be used to understand the performance of the leaned soft plan
- We obtain a notion of *context* for soft plans
- E.g. if the expectation in the current state is poor, we can fall back on another plan

Syntax `rl.expectedreturn(g, R)`

How a learning agent looks like

```
rl_parameter(policy, egreedy).
rl_parameter(alpha, 0.26).
rl_parameter(gamma, 0.9).
rl_parameter(epsilon, 0.22).
rl_parameter(epsilon_decay, 0.99992).

rl_observe(reach_finish, pos).

rl_reward(reach_finish, 10) :- finish_line.
rl_reward(reach_finish, -1) :- not finish_line.

rl_terminal(reach_finish) :- finish_line.

@action[rl_goal(reach_finish),
                   rl_param(direction(set(right, left, up, down)))]
+!move(Direction) <- move(Direction).

/* in this case, we run an infinite learning process - actually it
     could be stopped when the performance (expected return)
     is considered good enough */
!start.
+!start : true <- rl.execute(reach_finish); !start.
```

Extended Jason reasoning cycle



Extended Jason reasoning cycle details

- We extended the Jason architecture to include learning aspects
  Our additions are in red
    - (2b) Observations are updated
    - (7a-7b) The context of a plan can be bound to an expected return threshold
    - (11) rl.execute asks for the next action
    - (12) The RL reasoner gets all the needed information from the BB plus the relevant actions filtered by (7)
    - (13) rl.execute puts the next action on top of its intention
        - ▷ if the episode isn't over, another call of rl.execute is placed under the action in the same intention
- The next RL step is always performed after the execution of the previously selected action
- The RL reasoner is a black box for the agent
- The RL algorithm is a black box for the RL Coordinator

Key points

- The developer can inject domain knowledge through proper high-level abstractions
  - coherent way to manage states with beliefs
  - action set shaping through plans
- General approach independent from the RL algorithm – enables the use of the more fitting algorithm for the single task
- Multiple RL contexts/policies
- Learning tasks obtain modularity and reusability through soft plans
- Hierarchical approach thanks to plans (composability)

Next in line...

Case study

- Autonomous Driving as case study
  - To assess the advantages of the BDI-RL integration
- We expect benefits from both BDI and RL

The agent has to

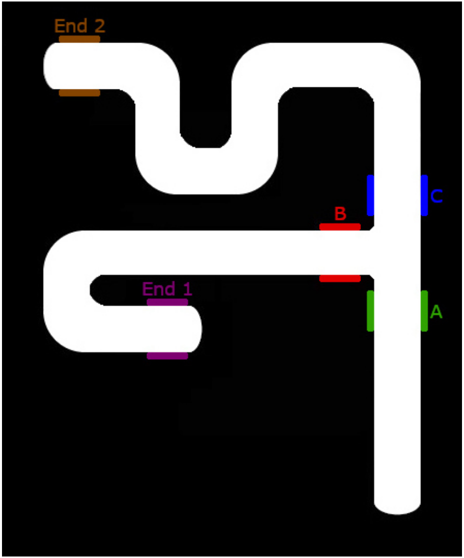
- follow high-level directions
- navigate without incidents

- RL struggles in *temporally extended planning* [Lake et al., 2017, Hassabis et al., 2017]
- Fine-grained navigation is an hard-to-engineer behavior

$\Rightarrow$ Hard-coded plans handle the high-level planning

$\Rightarrow$ Moving without incidents is achieved by RL

Intersection experiment

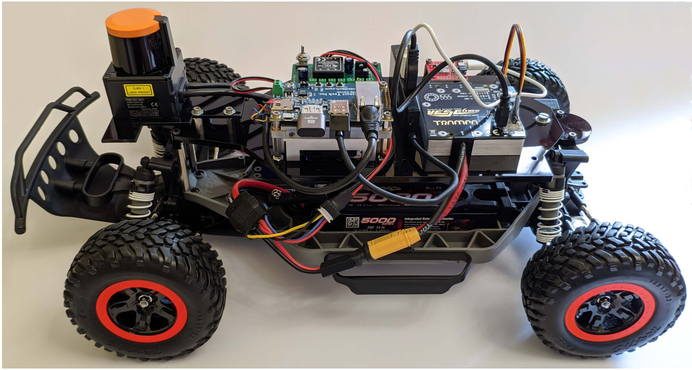


F1tenth platform I

The same code can run on:

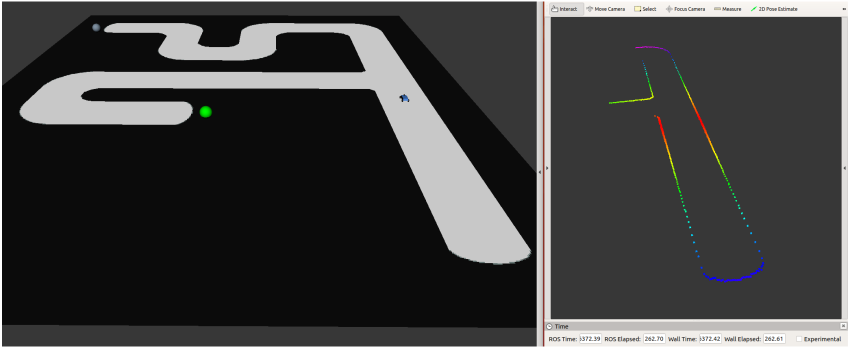
- A *very realistic* 1/10 scale car prototype
- The ad-hoc simulator

Hardware/software stacks similar to full-scale solutions



- Main experiment in the simulator
- Real car experiment in a simple track

F1tenth platform II



- ROS (Robot Operating System)
- Sensor node: Lidar, odometry
- Actuator node: VESC

Environment I

Control node

- Sensor data and actuator commands updated asynchronously
- Automatic emergency braking
  - It has priority over agent decisions
  - The car goes backward before returning the control to the agent

Percepts

- LIDAR vector `lidar_data(L)`
- Position label (thanks to odometry) `position(P)`
- Target label `target(T)`
- Emergency braking activated `crash`
- Position label has changed `new_position`
- Environment reward `reward(R)`

Environment II

Driving actions

- Go forward
- Turn right
- Turn left

⇒ Fast learning

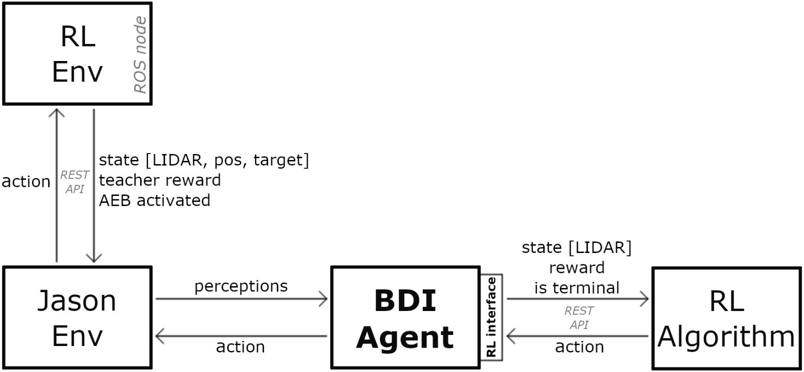
Environment actions

- Reset to position
- Ask for new target

Environment rewards

- Emergency braking = -1
- Proportional to
  - car velocity
  - distance to the nearest obstacle

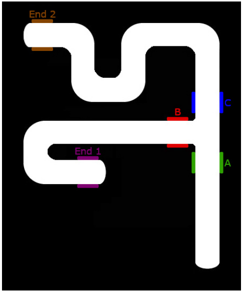
High-level modules



Next in line...

BDI agent I

- Two hard-plans that defines the high-level directions
  - They plan the moves to `END 1` and `END 2`
  - Defining sub-targets and self-rewards
- Three soft-plans that manages navigation in different situations
  - `follow_street`, `turn_left`, `go_forward`
  - Same code
  - Trained in different conditions and with different goals
  - ⇒ Distinct behaviors
- Four hard-plans for each soft-plans that handle the soft-plans outcome.
  - They govern the various situations and the learning cycle
    - ★ Emergency braking activation
    - ★ Wrong direction
    - ★ Target reached



BDI agent II

```
+target("END1") : true <-
    .println("");
    .println("target: END1");
    -+starting_point("START");
    -+target_point("A");
    !follow_street;
    .println("reached sub-target A");
    -+starting_point("A");
    -+target_point("B");
    !turn_left;
    .println("reached sub-target B");
    -+starting_point("B");
    -+target_point("END1");
    !follow_street;
    .println("reached target END1");
    .println("getting new target");
    reset_to_position("START");
    new_target.
```

```
+target("END2") : true <-
    .println("");
    .println("target: END2");
    -+starting_point("START");
    -+target_point("A");
    !follow_street;
    .println("reached sub-target A");
    -+starting_point("A");
    -+target_point("C");
    !go_forward;
    .println("reached sub-target C");
    -+starting_point("C");
    -+target_point("END2");
    !follow_street;
    .println("reached target END2");
    .println("getting new target");
    reset_to_position("START");
    new_target.
```

BDI agent III

```
+!follow_street : target_point(P) & position(P) <-
    move("stop");
    .println("reached ", P).

+!follow_street : starting_point(P) & position(P) <-
    rl.execute(follow_street);
    !follow_street.

+!follow_street : position("") <-
    rl.execute(follow_street);
    !follow_street.

+!follow_street : true <-
    move("stop");
    ?starting_point(P);
    .println("wrong direction taken");
    .println("resetting to point ", P);
    reset_to_position(P);
    !follow_street.
```

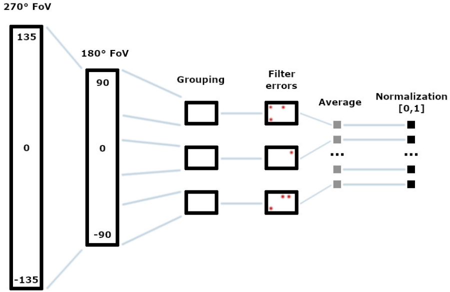
BDI agent IV

```
rl_algorithm(follow_street, dqn).
rl_observe(follow_street, lidar_data(list(1080))).
rl_terminal(follow_street) :- crash.
rl_terminal(follow_street) :- new_position.
rl_reward(follow_street, R) :- reward(R).
rl_reward(follow_street, 50) :- new_position & position(P) & target_point(P).
rl_reward(follow_street, -50) :- new_position & position(P)
                                   & not target_point(P) & not starting_point(P)
                                   & not position("").
```

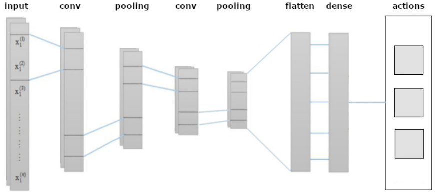
```
@action1[rl_goal(follow_street), rl_param(direction(set(forward, right, left)))]
+!move(Direction) <- move(Direction).
```

RL software I

- LIDAR pre-processing

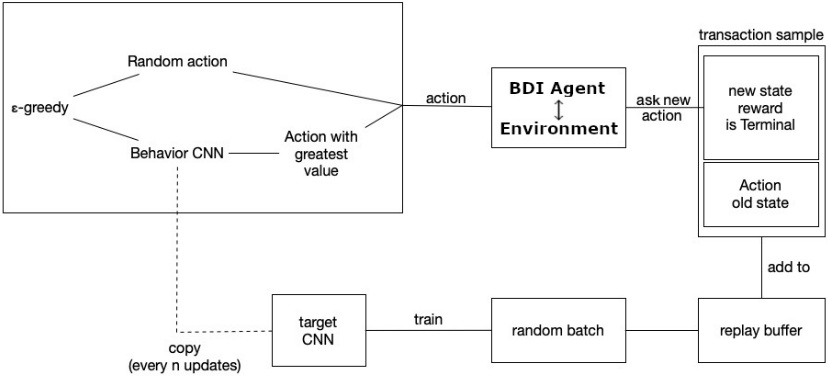


- Convolutional Neural Network



RL software II

- DQN
  - Standard enhancements: experience replay, state history, target net



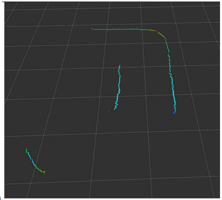
Next in line...

Intersection experiment results

- The agent can reach its target thanks to the integration of soft and hard plans
- Case study considerations
  - Problem that requires both time-extended planning and learned policies
  - Long-term plans are achieved thanks to the BDI reasoning
    - Hard-plans manage planning and soft-plan failures

- follow_street: orange
- turn_left: blue
- go_forward: red

Simulation to real



Use of LIDAR

- LIDAR measurements greatly affected by reflection [Ivanov et al., 2020]
- Training of RL agent with real LIDAR data is considered an *open problem*

Continuous time

Appropriate cycle timing

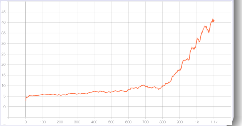
Reset Mechanism

Applicable in the real-world

Results and comparison

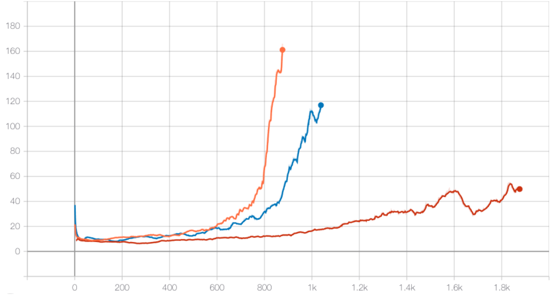
Real car results

- The driver-agent successfully learned a control policy
  - using real LIDAR data
- 1D CNN used to process LIDAR data for the first time



NNs comparison

- CNNs perform better on structured and spatially related data
- 1D CNNs are very effective in processing LIDAR data



- 1D CNN: orange
- Dense: blue
- 2D CNN: red

Next in line...

Discussion I

Agentspeak

- Imperative nature
- Declarative nature

Automating the development of the imperative side

- Inclusion of soft-plans in the reasoning flow of the agent
- Manage and use learning in a declarative way with powerful abstractions
- Preserving the reasoning capabilities of a BDI agent
  - The framework act on the plan’s body
  - It retains the declarative attitude derived from event triggering and plans’ context

Multiple RL contexts

- Coordinated inside the same agent
- To achieve complex and time-extended tasks

Discussion II

Case study considerations

- Mixing hard and soft plans
- Problem that requires both time-extended planning and learned policies
- Long-term plans are achieved thanks to the BDI reasoning
- Hard-plans manage planning and soft-plan failures

Advantages wrt end-to-end RL

- RL struggles in temporally-extended planning [Lake et al., 2017, Hassabis et al., 2017]
  - A plain RL agent will need more training time
  - It may not be able to learn such behavior at all
  - This is even more true if we consider broader scenarios (in the BDI-RL agent, we can simply add a few soft-plans)
- BDI reasoning is effective to achieve planning and coordination of learned policies
- BDI abstractions allow to split and enhance RL signals among different contexts
- Use of the most appropriate RL algorithm for each sub-task
  - As multiple RL algorithms can coexist

Future works I

Next steps

- Intersection experiment with the real f1tenth
- BDI-RL vs plain RL *quantitative* comparison

Engineering

- Explore further the education process lifecycle and stages relations
- Develop proper tools to be embedded in existing IDEs
- Exploring software engineering aspects
- Verify what is the impact on AOSE (Agent-Oriented Software Engineering)

Agent systems

- Consider first-class abstractions such as artifacts in the process
- Consider multi-agent systems and agent organizations
- Cooperative learning in MAS
  - exchange of experience with efficient communication [Kamp et al., 2018]

Future works II

RL extensions

- Use of compound actions (option framework)
- Hierarchical RL – it allows to aggregate actions into reusable subroutines
- Reward Shaping – “education” through demonstrations
- Continuous action space RL
- Meta-learning
- Reinforcement learning with bounded risk [Geibel, 2001]

RL extensions

- Urban scenario with directions given by a maps service
  - Using advanced simulators like Carla [Dosovitskiy et al., 2017]
- From 1/10 to real vehicles

References

- Reinforcement Learning [Sutton and Barto, 2018]
- Agent-Oriented Programming [Shoham, 1993]
- BDI [Rao and Georgeff, 1995]
- Reasoning Cycle [Wooldridge, 2009]
- Jason [Bordini et al., 2007]
- HRL [Botvinick et al., 2009]
- Shaping in RL [Brys et al., 2015]
- Agents&Artifacts [Ricci et al., 2011]

References I

- Airiau, S., Padgham, L., Sardina, S., and Sen, S. (2009).
  Enhancing the adaptation of BDI agents using learning techniques.
  Int. J. Agent Technol. Syst., 1(2):1–18.

- Arpteg, A., Brinne, B., Crnkovic-Friis, L., and Bosch, J. (2018).
  Software engineering challenges of deep learning.
  In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 50–59.

- Badica, A., Badica, C., Ivanovic, M., and Mitrovic, D. (2015).
  An approach of temporal difference learning using agent-oriented programming.
  In *20th Int. Conf. on Control Systems and Computer Science*, pages 735–742.

- Badica, C., Becheru, A., and Felton, S. (2017).
  Integration of jason reinforcement learning agents into an interactive application.
  In *19th Int. Symp. on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 361–368.

References II

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007).
Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology).
John Wiley & Sons, Inc., USA.

Botvinick, M. M., Niv, Y., and Barto, A. C. (2009).
Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective.
Cognition, 113(3):262 – 280.
Reinforcement learning and higher cognition.

Brys, T., Harutyunyan, A., Suay, H. B., Chernova, S., Taylor, M. E., and Nowé, A. (2015).
Reinforcement learning from demonstration through shaping.
In *Proc. of the 24th Int. Conf. on Artificial Intelligence (IJCAI '15)*, IJCAI'15, pages 3352–3358. AAAI Press.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017).
CARLA: An open urban driving simulator.
In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16.

References III

Feliú, J. L. S. (2013).
Use of Reinforcement Learning (RL) for plan generation in Belief-Desire-Intention (BDI) agent systems.
University of Rhode Island.

Geibel, P. (2001).
Reinforcement learning with bounded risk.
In *ICML*, pages 162–169.

Guerra-Hernández, A., El Fallah-Seghrouchni, A., and Soldano, H. (2004).
Learning in BDI multi-agent systems.
In *Proc. of the 4th Int. Conf. on Computational Logic in Multi-Agent Systems (CLIMA VI)*, CLIMA IV'04, pages 218–233, Berlin, Heidelberg. Springer-Verlag.

Hassabis, D., Kumaran, D., Summerfield, C., and Botvinick, M. (2017).
Neuroscience-inspired artificial intelligence.
Neuron, 95:245–258.

References IV

Ivanov, R., Carpenter, T. J., Weimer, J., Alur, R., Pappas, G. J., and Lee, I. (2020).
Case study: Verifying the safety of an autonomous racing car with a neural network controller.
In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, HSCC '20, New York, NY, USA. Association for Computing Machinery.

Jones, N. (2014).
The learning machines.
Nature, 505(7482):146.

Kamp, M., Adilova, L., Sicking, J., Hüger, F., Schlicht, P., Wirtz, T., and Wrobel, S. (2018).
Efficient decentralized deep learning by dynamic model averaging.

Karim, S., Sonenberg, L., and Tan, A.-H. (2006).
A hybrid architecture combining reactive plan execution and reactive learning.
In Yang, Q. and Webb, G., editors, *PRICAI 2006: Trends in Artificial Intelligence*, pages 200–211, Berlin, Heidelberg. Springer Berlin Heidelberg.

References V

Karpathy, A. (2017).
Software 2.0.

Khomh, F., Adams, B., Cheng, J., Fokaefs, M., and Antoniol, G. (2018).
Software engineering for machine-learning applications: The road ahead.
IEEE Software, 35(5):81–84.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017).
Building machines that learn and think like people.
Behavioral and Brain Sciences, 40:e253.

Meijer, E. (2018).
Behind every great deep learning framework is an even greater programming languages concept (keynote).
In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, pages 1–1, New York, NY, USA. ACM.

References VI

Norling, E. (2004).
Folk psychology for human modelling: Extending the bdi paradigm.
In *Proc. of the Third Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS '04)*, AAMAS '04, pages 202–209, Washington, DC, USA. IEEE Computer Society.

Rao, A. S. and Georgeff, M. P. (1995).
BDI agents: From theory to practice.
In *Proc. of the First Int. Conf. on Multi-Agent Systems (ICMAS-95*, pages 312–319.

Ricci, A., Piunti, M., and Viroli, M. (2011).
Environment programming in multi-agent systems: an artifact-based perspective.
Autonomous Agents and Multi-Agent Systems, 23(2):158–192.

Shoham, Y. (1993).
Agent-oriented programming.
Artif. Intell., 60(1):51–92.

References VII

Singh, D. and Hindriks, K. V. (2013).
Learning to improve agent behaviours in goal.
In Dastani, M., Hübner, J. F., and Logan, B., editors, *Programming Multi-Agent Systems*, pages 158–173, Berlin, Heidelberg. Springer Berlin Heidelberg.

Singh, D., Sardina, S., Padgham, L., and James, G. (2011).
Integrating learning into a BDI agent for environments with changing dynamics.
In *Proc. of the Twenty-Second Int. Joint Conf. on Artificial Intelligence (IJCAI '11)*, IJCAI'11, pages 2525–2530. AAAI Press.

Sutton, R. S. and Barto, A. G. (2018).
Reinforcement learning : an introduction.
The MIT Press.

Tan, A.-H., Ong, Y.-S., and Tapanuj, A. (2011).
A hybrid agent architecture integrating desire, intention and reinforcement learning.
Expert Syst. Appl., 38(7):8477–8487.

References VIII

Weiß, G. (1996).
Adaptation and learning in multi-agent systems: Some remarks and a bibliography.
In Weiß, G. and Sen, S., editors, *Adaption and Learning in Multi-Agent Systems*, pages 1–21, Berlin, Heidelberg. Springer Berlin Heidelberg.

Wooldridge, M. (2009).
Introduction to Multi-Agent Systems.
Wiley.

Integrating BDI and Reinforcement Learning

Michael Bosello Alessandro Ricci Giovanni Pau

Alma Mater Studiorum – Università di Bologna
Department of Computer Science and Engineering, Cesena Campus

michael.bosello@unibo.it

- From Programming Agents to *Educating* Agents
  A Jason-based Framework for Integrating Learning in the Development of Cognitive Agents
- Integrating BDI and Reinforcement Learning: the Case Study of Autonomous Driving