TorchCraft: a Library for Machine Learning Research on Real-Time Strategy Games

Gabriel Synnaeve, Nantas Nardelli, Alex Auvolat, Soumith Chintala,
Timothée Lacroix, Zeming Lin, Florian Richoux, Nicolas Usunier
gab@fb.com, nantas@robots.ox.ac.uk

November 3, 2016

Abstract

We present TorchCraft, an open-source library that enables deep learning research on Real-Time Strategy (RTS) games such as StarCraft: Brood War, by making it easier to control these games from a machine learning framework, here Torch [9]. This white paper argues for using RTS games as a benchmark for AI research, and describes the design and components of TorchCraft.

1 Introduction

Deep Learning techniques [13] have recently enabled researchers to successfully tackle low-level perception problems in a supervised learning fashion. In the field of Reinforcement Learning this has transferred into the ability to develop agents able to learn to act in high-dimensional input spaces. In particular, deep neural networks have been used to help reinforcement learning scale to environments with visual inputs, allowing them to learn policies in testbeds that previously were completely intractable. For instance, algorithms such as Deep Q-Network (DQN) [14] have been shown to reach human-level performances on most of the classic ATARI 2600 games by learning a controller directly from raw pixels, and without any additional supervision beside the score. Most of the work spawned in this new area has however tackled environments where the state is fully observable, the reward function has no or low delay, and the action set is relatively small. To solve the great majority of real life problems agents must instead be able to handle partial observability, structured and complex dynamics, and noisy and high-dimensional control interfaces.

To provide the community with useful research environments, work was done towards building platforms based on videogames such as Torcs [27], Mario AI [20], Unreal's BotPrize [10], the Atari Learning Environment [3], VizDoom [12], and Minecraft [11], all of which have allowed researchers to train deep learning models with imitation learning, reinforcement learning and various decision making algorithms on increasingly difficult problems. Recently there have also been efforts to unite those and many other such environments in one platform to provide a standard interface for interacting with them [4]. We propose a bridge between StarCraft: Brood War, an RTS game with an active AI research community and annual AI competitions [16, 6, 1], and Lua, with examples in Torch [9] (a machine learning library).

2 Real-Time Strategy for Games AI

Real-time strategy (RTS) games have historically been a domain of interest of the planning and decision making research communities [5, 2, 6, 16, 17]. This type of games aims to simulate the control of multiple units in a military setting at different scales and level of complexity, usually in a fixed-size 2D map, in duel or in small teams. The goal of the player is to collect resources which can be used to expand their control on the map, create buildings and units to fight off enemy deployments, and ultimately destroy the opponents. These games exhibit durative moves (with complex game dynamics) with simultaneous actions (all players can give commands to any of their units at any time), and very often partial observability (a "fog of war": opponent units not in the vicinity of a player's units are not shown).

RTS gameplay: Components RTS game play are economy and battles ("macro" and "micro" respectively): players need to gather resources to build military units and defeat their opponents. To that end, they often have worker units (or extraction structures) that can gather resources needed to build workers, buildings, military units and research upgrades. Workers are often also builders (as in StarCraft), and are weak in fights compared to military units. Resources may be of varying degrees of abundance and importance. For instance, in StarCraft minerals are used for everything, whereas gas is only required for advanced buildings or military units, and technology upgrades. Buildings and research define technology trees (directed acyclic graphs) and each state of a "tech tree" allow for the production of different unit types and the training of new unit abilities. Each unit and building has a range of sight that provides the player with a view of the map. Parts of the map not in the sight range of the player's units are under fog of war and the player cannot observe what happens there. A considerable part of the strategy and the tactics lies in which armies to deploy and where.

Military units in RTS games have multiple properties which differ between unit types, such as: attack range (including melee), damage types, armor, speed, area of effects, invisibility, flight, and special abilities. Units can have attacks and defenses that counter each others in a rock-paper-scissors fashion, making planning armies a extremely challenging and strategically rich process. An "opening" denotes the same thing as in Chess: an early game plan for which the player has to make choices. That is the case in Chess because one can move only one piece at a time (each turn), and in RTS games because, during the development phase, one is economically limited and has to choose which tech paths to pursue. Available resources constrain the technology advancement and the number of units one can produce. As producing buildings and units also take time, the arbitrage between investing in the economy, in technological advancement, and in units production is the crux of the strategy during the whole game.

Related work: Classical AI approaches normally involving planning and search [2, 15, 24, 7] are extremely challenged by the combinatorial action space and the complex dynamics of RTS games, making simulation (and thus Monte Carlo tree search) difficult [8, 22]. Other characteristics such as partial observability, the non-obvious quantification of the value of the state, and the problem of featurizing a dynamic and structured state contribute to making them an interesting problem, which altogether ultimately also make them an excellent benchmark for AI. As the scope of this paper is not to give a review of RTS AI research, we refer the reader to these surveys about existing research on RTS and StarCraft AI [16, 17].

It is currently tedious to do machine learning research in this domain. Most previous reinforcement learning research involve simple models or limited experimental settings [26, 23]. Other models are trained on offline datasets of highly skilled players [25, 18, 19, 21]. Contrary to most Atari games [3], RTS games have much higher action spaces and much more structured states. Thus, we advocate here to have not only the pixels as input and keyboard/mouse for commands, as in [3, 4, 12], but also a structured representation of the game state, as in

```
-- main game engine loop:
while true do
    game.receive_player_actions()
    game.compute_dynamics()
    -- our injected code:
    torchcraft.send_state()
    torchcraft.receive_actions()
end
```

```
featurize, model = init()
tc = require 'torchcraft'
tc:connect(port)
while not tc.state.game_ended do
    tc:receive()
    features = featurize(tc.state)
    actions = model:forward(features)
    tc:send(tc:tocommand(actions))
end
```

Figure 1: Simplified client/server code that runs in the game engine (server, on the left) and the library for the machine learning library or framework (client, on the right).

[11]. This makes it easier to try a broad variety of models, and may be useful in shaping loss functions for pixel-based models.

Finally, StarCraft: Brood War is a highly popular game (more than 9.5 million copies sold) with professional players, which provides interesting datasets, human feedback, and a good benchmark of what is possible to achieve within the game. There also exists an active academic community that organizes AI competitions.

3 Design

The simplistic design of TorchCraft is applicable to any video game and any machine learning library or framework. Our current implementation connects Torch to a low level interface [1] to StarCraft: Brood War. TorchCraft's approach is to dynamically inject a piece of code in the game engine that will be a server. This server sends the state of the game to a client (our machine learning code), and receives commands to send to the game. This is illustrated in Figure 1. The two modules are entirely synchronous, but the we provide two modalities of execution based on how we interact with the game:

Game-controlled - we inject a DLL that provides the game interface to the bots, and one that includes all the instructions to communicate with the machine learning client, interpreted by the game as a player (or bot AI). In this mode, the server starts at the beginning of the match and shuts down when that ends. In-between matches it is therefore necessary to re-establish the connection with the client, however this allows for the setting of multiple learning instances extremely easily.

Game-attached - we inject a DLL that provides the game interface to the bots, and we interact with it by attaching to the game process and communicating via pipes. In this mode there is no need to re-establish the connection with the game every time, and the control of the game is completely automatized out of the box, however it's currently impossible to create multiple learning instances on the same guest OS.

Whatever mode one chooses to use, TorchCraft is seen by the AI programmer as a library that provides: `connect()`, `receive()` (to get the state), `send(commands)`, and some helper functions about specifics of StarCraft's rules and state representation. TorchCraft also provides an efficient way to store game frames data from past (played or observed) games so that existing state ("replays", "traces") can be re-examined.

4 Conclusion

We presented several work that established RTS games as a source of interesting and relevant problems for the AI research community to work on. We believe that an efficient bridge between low level existing APIs and machine learning frameworks/libraries would enable and foster research on such games. We presented TorchCraft: a library that enables state-of-the-art machine learning research on real game data by interfacing Torch with StarCraft: BroodWar. TorchCraft has already been used in reinforcement learning experiments on StarCraft, which led to the results in [23] (soon to be open-sourced too and included within TorchCraft).

5 Acknowledgements

We would like to thank Yann LeCun, Léon Bottou, Pushmeet Kohli, Subramanian Ramamoorthy, and Phil Torr for the continuous feedback and help with various aspects of this work. Many thanks to David Churchill for proofreading early versions of this paper.

References

[1] BWAPI: Brood war api, an api for interacting with starcraft: Broodwar (1.16.1). `https://bwapi.github.io/`, 2009–2015.

[2] Aha, D. W., Molineaux, M., and Ponsen, M. Learning to win: Case-based plan selection in a real-time strategy game. In *International Conference on Case-Based Reasoning* (2005), Springer, pp. 5–20.

[3] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* (2012).

[4] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[5] Buro, M., and Furtak, T. Rts games and real-time ai research. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)* (2004), vol. 6370.

[6] Churchill, D. Starcraft ai competition. `http://www.cs.mun.ca/~dchurchill/starcraftaicomp/`, 2011–2016.

[7] Churchill, D. *Heuristic Search Techniques for Real-Time Strategy Games*. PhD thesis, University of Alberta, 2016.

[8] Churchill, D., Saffidine, A., and Buro, M. Fast heuristic search for rts game combat scenarios. In *AIIDE* (2012).

[9] Collobert, R., Kavukcuoglu, K., and Farabet, C. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop* (2011), no. EPFL-CONF-192376.

[10] Hingston, P. A turing test for computer game bots. *IEEE Transactions on Computational Intelligence and AI in Games 1*, 3 (2009), 169–186.

[11] Johnson, M., Hofmann, K., Hutton, T., and Bignell, D. The malmo platform for artificial intelligence experimentation. In *International joint conference on artificial intelligence (IJCAI)* (2016).

[12] Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. Vizdoom: A doom-based ai research platform for visual reinforcement learning. *arXiv preprint arXiv:1605.02097* (2016).

[13] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature 521*, 7553 (2015), 436–444.

[14] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (2015), 529–533.

[15] Ontañón, S., Mishra, K., Sugandh, N., and Ram, A. Case-based planning and execution for real-time strategy games. In *International Conference on Case-Based Reasoning* (2007), Springer Berlin Heidelberg, pp. 164–178.

[16] Ontanón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., and Preuss, M. A survey of real-time strategy game ai research and competition in starcraft. *Computational Intelligence and AI in Games, IEEE Transactions on 5*, 4 (2013), 293–311.

[17] Robertson, G., and Watson, I. A review of real-time strategy game ai. *AI Magazine 35*, 4 (2014), 75–104.

[18] Synnaeve, G. *Bayesian programming and learning for multi-player video games: application to RTS AI*. PhD thesis, PhD thesis, Institut National Polytechnique de Grenoble—INPG, 2012.

[19] Synnaeve, G., and Bessiere, P. A dataset for starcraft ai & an example of armies clustering. *arXiv preprint arXiv:1211.4552* (2012).

[20] Togelius, J., Karakovskiy, S., and Baumgarten, R. The 2009 mario ai competition. In *IEEE Congress on Evolutionary Computation* (2010), IEEE, pp. 1–8.

[21] Uriarte, A. Starcraft brood war data mining. `http://nova.wolfwork.com/dataMining.html`, 2015.

[22] Uriarte, A., and Ontañón, S. Game-tree search over high-level game states in rts games. In *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference* (2014).

[23] Usunier, N., Synnaeve, G., Lin, Z., and Chintala, S. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *arXiv preprint arXiv:1609.02993* (2016).

[24] Weber, B. Reactive planning for micromanagement in rts games. *Department of Computer Science, University of California, Santa Cruz* (2014).

[25] Weber, B. G., and Mateas, M. A data mining approach to strategy prediction. In *2009 IEEE Symposium on Computational Intelligence and Games* (2009), IEEE, pp. 140–147.

[26] Wender, S., and Watson, I. Applying reinforcement learning to small scale combat in the real-time strategy game starcraft: broodwar. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on* (2012), IEEE, pp. 402–408.

[27] Wymann, B., Espié, E., Guionneau, C., Dimitrakakis, C., Coulom, R., and Sumner, A. Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net* (2000).

A Frame data

In addition to the visual data, the TorchCraft server extracts certain information for the game state and sends it over to the connected clients in a structured "frame". The frame is formatted in a table in roughly the following structure:

```
Received update:  {
 // Number of frames in the current game
 // NB: a 'game' can be composed of several battles
 frame_from_bwapi : int
  units_myself :
   {
   // Unit ID
   int:
    {
    // Unit ID
    target : int
    targetpos :
     {
     // Absolute x
     1 : int
     // Absolute y
     2 : int
     }
    // Type of air weapon
    awtype : int
    // Type of ground weapon
    gwtype : int
    // Number of frames before next air weapon possible attack
    awcd : int
    // Number of hit points
    hp : int
    // Number of energy / mana points, if any
    energy : int
    // Unit type
    type : int
    position :
     {
     // Absolute x
     1 : int
     // Absolute y
     2 : int
     }
    // Number of armor points
    armor : int
    // Number of frames before next ground weapon possible attack
    gwcd : int
    // Ground weapon attack damage
    gwattack : int
    // Protoss shield points (like HP, but with special properties)
    shield : int
    // Air weapon attack damage
    awattack : int (air weapon attack damage)
    // Size of the unit
    size : int
    // Whether unit is an enemy or not
    enemy : bool
    // Whether unit is idle, i.e. not following any orders currently
    idle : bool
    // Ground weapon max range
    gwrange : int
    // Air weapon max range
    awrange : int
   }
  }
  // Same format as "units_myself"
  units_enemy : ...
 }
```