Neural MMO: Ingredients for Massively Multiagent Artificial Intelligence Research

Joseph Suarez [1] **Phillip Isola** [1]

Abstract

Simulated games have become a staple of multiagent intelligence research. Our work considers massively multiplayer online role-playing games (MMORPGs or MMOs), a genre of games that has only recently begun to gain attention in the reinforcement learning community. MMOs capture several complexities of real-world learning that are difficult to integrate with existing methods. We present several highlights from the ongoing development of Neural MMO that are particularly relevant to enabling reinforcement learning methods in artificial open worlds.

1. Introduction

From arcade to first-person shooter (FPS) to real-time strategy (RTS) and massive online battle arena (MOBA), increasingly complex game environments have accelerated recent progress in deep reinforcement learning (RL) (Silver et al., 2016; Baker et al., 2019; OpenAI, 2018; Berner et al., 2019; Vinyals et al., 2019; Jaderberg et al., 2018). MMOs simulate self-contained macrocosms with large, variable numbers of players and realistic social strategy. Neural MMO is not yet a full scale MMO but takes several steps in this direction.

We release convolutional, attentional, and recurrent baseline policies [Video] capable of robust foraging and combat over thousands of timesteps. All baselines make use of an efficient IO subarchitecture that allows agents to interface with Neural MMO's rich underlying observation and action spaces. Policies are trained with PPO (Schulman et al., 2017) + GAE (Schulman et al., 2015) with no reward shaping or other modifications. Agent value and attention functions have learned reasonable estimates of the significance and utility of different tiles (Section 3). Results are reproducible overnight on personal-grade hardware.

[1]Massachusetts Institute of Technology. Correspondence to: Joseph Suarez <jsuarez@mit.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

2. Neural MMO

Neural MMO is a massively multiagent environment for artificial intelligence research. Agents forage for resources and engage in strategic combat within a persistent game world . Our environment implements a progression system inspired by traditional MMOs and a full 3D renderer for visualizations. Figure 1 details the three core game systems. Agents may move aroung the grass and forest tiles of the game map, but stone and water is impassible; lava is lethal.

Resource System: Agents spawn with 10 (configurable) units of food, water, and health. At every time step, agents lose 1 food and 1 water. If agents run out of food or water, they begin losing health. If agents are well fed and well hydrated, they begin regaining health. In order to survive, agents must quickly forage for food, which is in limited supply, and water, which is infinitely renewable but only available at a smaller number of pools, in the presence of 100+ potentially hostile agents attempting to do the same.

Combat System: Agents can attack each other with three different styles – Range, Mage, and Melee. Accuracy and damage are determined by the attack style and the combat stats of the attacker and defender. This system enables a variety of strategies. Agents more skilled in combat can assert map control, locking down resource rich regions for themselves. Agents more skilled in maneuvering can succeed through foraging and evasion. The goal is to balance between foraging safely and engaging in dangerous combat to pilfer other agents' resources and cull the competition.

Progression System: Progress in real MMOs varies on two axes: soft advantage gained through strategic/mechanical talent and hard numerical advantage gained through skill levels/equipment. In Neural MMO, agents progress their abilities through usage. Foraging for food and water grants experience in the respective Hunting and Fishing skills, which enable agents to gather and carry more resources. A similar system is in place for combat. Agents gain levels in Constitution, Range, Mage, Melee, and Defense through fighting other agents. Higher offensive levels increase accuracy and damage while Constitution and Defense increase health and evasion.

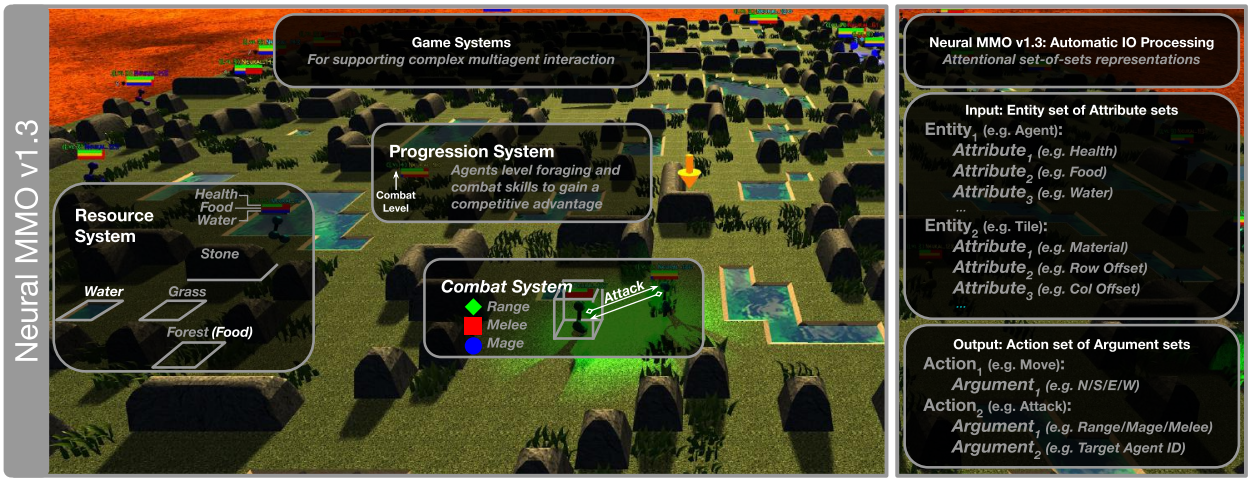

Figure 1. Neural MMO is a massively multiagent environment for AI research. Agents compete for resources through foraging and combat. Observation and action representation in local game state enable efficient training and inference. A 3D Unity client provides high quality visualizations for interpreting learned behaviors. The environment, client, training code, and policies are fully open source, officially documented, and actively supported through a live community Discord server.

3. Experiments: RLlib Baseline Models

Neural MMO is a large environment with variably sized populations and complex structured IO (observation/action) spaces. These properties are essential to the development of Neural MMO as an increasingly rich artificial open world. However, existing reinforcement learning frameworks were not designed with these properties in mind.

RLlib (Liang et al., 2017) is a popular reinforcement learning framework built on top of the excellent Ray (Moritz et al., 2018) distributed computing library. We have worked with the developers to add support at the framework level: RLlib and Neural MMO are now natively compatible. As has been the case in recent work on large multiagent environments, training quality policies in Neural MMO became quite straightforward given good infrastructure. RLlib's default PPO implementation coupled with a small baseline model produces high-quality policies capable of surviving for thousands of timesteps (Figure 2). Our networks require only a few hours of training on 4 CPU cores and a single GTX 1080 TI GPU at 10 percent utilization.

In the remainder of this paper, we visualize emergent intelligent properties of the learned policy and discuss key properties of the architecture and environment design. We strongly encourage readers to watch the anonymized two minute video linked in the Introduction.

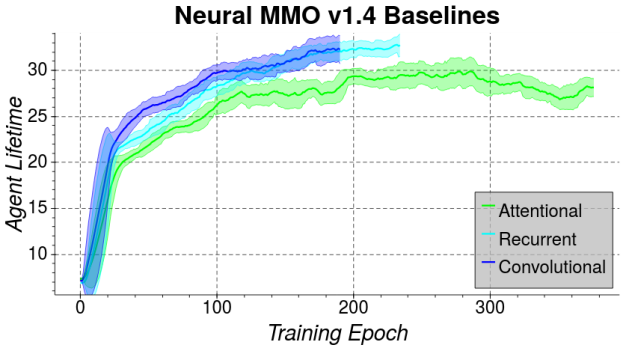

Figure 2. Baseline training curves. Recurrent and convolutional policies both perform quite well; agents typically die in unpredictable combat rather than from obvious foraging errors.

4. In-Build Research Overlays

Neural MMO includes a full Unity3D client for test-time rendering and associated research tools that enable users to visualize various properties of learned agent policies. In addition to viewing the progression levels and behaviors of individual agents, users can register custom overlays – effectively image maps rendered over the environment with transparency and updated in real time. Our API enables users to render arbitrary custom overlays; the four shown in Figure 4 are included with the baseline policies.

The counts exploration overlay shows that agents spread out to cover the entire map, as well as that certain areas are more popular than others. Value function overlays demonstrate that agents have correctly learned the importance of being near both food and water, not just either individually. Likewise, tiles near the edge of the map are unfavorable, as this is where competing agents spawn. We can also see that agents have correctly learned to attend to resources (food and water) and obstacles in their path while ignoring irrelevant grass tiles.

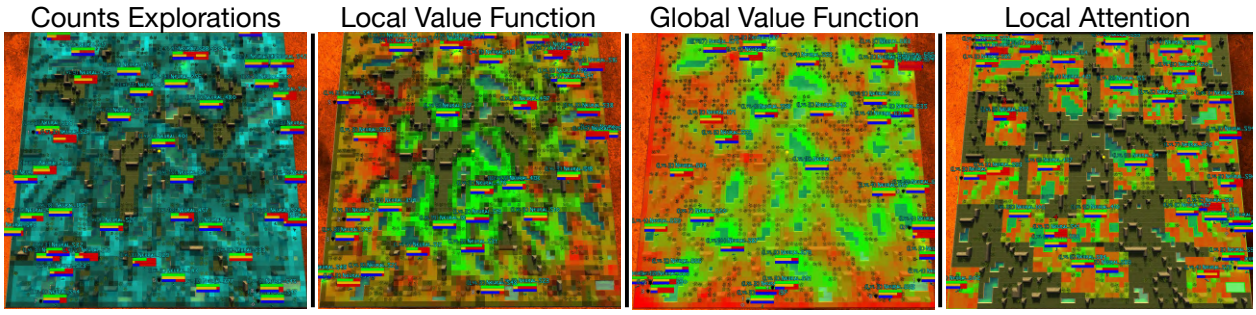

Figure 3. In-game overlays visualize various aspects of learned policies. Counts exploration shades tiles based on the number of visitations. Baseline policies explore all but a few small and resource-poor areas of the map. Local and global value functions shade tiles based on agents' learned value function. The local value overlay shades tiles as agents walk over them with the current value function, taking into account the current state of the environment. The global value function is precomputed for each tile as if no other agents were present. In either case, agents have learned to value adjacent clumps of food and water. The attention overlay shows which tiles most influence agent actions. See Figure 4 for additional information on this overlay.

5. Environment Design

5.1. Meaningful Persistence

The systems described in Section 2 encourage robust strategy over long horizons of persistent play with no environment resets. We also confirm that some such behaviors are learned in practice. The Convolutional and Recurrent baselines policies both develop robust foraging and combat strategies. We have seen individual agents survive for up to 10,000 timesteps – around an hour and a half in real time.

Not all persistence is meaningful. If an agent can survive for 10,000 timesteps by repeatedly solving five- or ten-step planning problems to navigate to the next food pellet, then actions at time step 50 have practically no influence on outcomes at time step 10,000. That is, the environment is persistent in name only. In Neural MMO, meaningful persistence is introduced through strong interagent dependencies – agents must develop reasoning over increasingly long time horizons to have a fighting chance at survival. To gain a competitive advantage, agents are incentivized to level their foraging and combat stats efficiently, taking smart risks to secure resource-rich terrain and eliminate vulnerable adversaries. Inefficient agents will find themselves disadvantaged when they come into contact with higher level opponents.

5.2. Randomization

Early in development, Neural MMO used hand-crafted game maps with no randomization. Learning was difficult and inconsistent at best, even at substantial scale. Recent works (Cobbe et al., 2019; OpenAI et al., 2019) have found that environment randomization serves as an effective learning curriculum. In Neural MMO, training on 256 randomly generated maps significantly increases learnability and final policy quality. Including the test map as one of the 256 actually results in higher policy quality than training directly and exclusively on the test map. (Cobbe et al., 2019) suggest that even more maps may be beneficial.

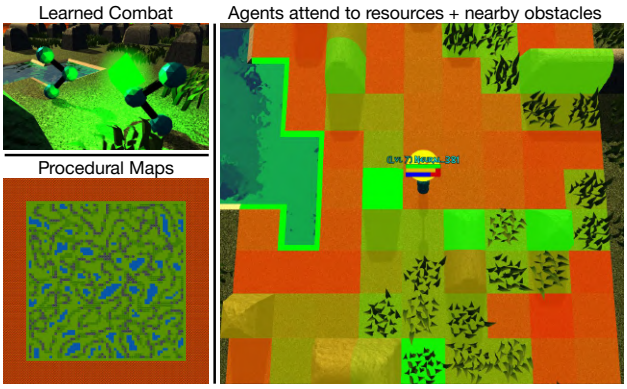

Figure 4. Top left: agents learn to engage in opportunistic combat, pilfering resources from nearby agents. Bottom left: procedural map generation creates a smooth learning curriculum and improves policy robustness. Right: Agents attend to food (below) + water and nearby obstacles (left) but ignore grass tiles (above/right).

5.3. Efficient Internal Representation

In Neural MMO, we procedurally generate tile-based terrain by thresholding 2D Perlin ridge fractals (Perlin, 1985). Environments simulating realistic physics are often preferred over more "toy" grid-worlds. Physically simulated environments do have their place, especially in robotics and real-world transfer tasks. However, it is often possible to achieve far greater complexity per unit of compute in grid-worlds. This approach has proven effective: some of the most popular and comprehensive open-world MMOs actually use a grid representation internally – animation smoothing keeps gameplay responsive while keeping servers lightweight.

6. General IO

Small scale RL environments typically provide input observations as raw data tensors and output actions as low-dimensional vectors. More complex environments may contain variable length observation and action spaces with mixed data types: standard architectures that expect fixed length tensors cannot process these IO spaces. Our solution to this problem parameterizes the observation space as a set of entities, each of which is parameterized by a set of attributes (Figure 1, Input). We automatically generate attentional networks to select variable length action arguments by keying against learned entity embeddings (Figure 1, Output).

6.1. The Input Problem

We define local game state by the set of observable objects or *entities* (e.g. agents and tiles), each of which is parameterized by a number of local properties or *attributes* (e.g. health, food, water). At compile time, embedding networks $e_{x_1}, e_{x_2}, ...$ are defined for each attribute. We also define soft attention functions $\{f_{y_j}\}$ and g to be used later.

Input	*Entity set of attribute sets*
$X \coloneqq \{x_i\}$	Define attributes x_i
$Y = \{e_{x_i}(x_i)\}$	Embed entity attributes
$Z = \{f_{y_j}(\{x_{1,...,n}\} \subseteq Y)\}$	Soft attend f_{y_i} to attributes
$o = g(Z)$	Soft attend g to entities

At run time, we project $x_1, x_2, ...$ to fixed length attribute embedding vectors $y_1, y_2, ...$ using embedding layers e_{x_i}. Soft attention layers f_{y_j} aggregrate across the attribute embeddings of each entity to produce a representation z_i for each entity. Finally, an attentional layer g aggregates across all entity embeddings Z to produce a flat observation embedding o. We return both o and Z.

6.2. The Output Problem

We define agent decision space by a list of actions, each of which takes a list of arguments. Actions are callable function references that the environment can invoke on the associated argument list in order to execute an agent's decision, such as Move → [North] or Attack → [Melee, Agent ID]. At compile time, the user specifies a hard attentional architecture h. We reuse the Input module to generate embedding layers for all arguments.

Output	*Action list of arg lists*
$A \coloneqq [A_i : [a_1, ..., a_n], ...]$	Define arguments A_{ij}
$B_{ij} = e_{a_{ij}}(A_{ij})$	Embed arguments to B_{ij}
$\text{arg}_{A_i} = h(o, \{B_i, \widetilde{z}_i \subseteq Z\})$	Hard attend f to args

At run time, we convert the hidden state o of the main network into an action list of argument lists. To do so, we embed candidate arguments for all actions A_{ij} to fixed length vectors B_{ij} using $e_{a_{ij}}$, similarly to as in the Input module. As entities can be arguments, we will also consider Z from the input network. For each candidate action-argument A_{ij}, we compare embeddings $\{B_i, \widetilde{z}_i\}$ to the hidden state using the attentional function h to produce a softmax distribution. Sampling from this distribution yields a hard attentional choice arg_{A_i} over arguments. Finally, we return game actions paired with their selected argument lists.

7. Discussion and Limitations

Comparison to OpenAI Five: Neural MMO and OpenAI Five (OpenAI, 2018) are both many-agent systems trained entirely via self-play using PPO. OpenAI Five uses reward shaping to solve DoTA 2, a fixed 5v5 task. The resultant policies are much more complex than those learned in Neural MMO and are capable of beating top professional players. By comparison, Neural MMO operates over large (100+) variably sized agent populations using $< 0.001\%$ of OpenAI Five scale hardware and sparse, unshaped rewards.

IO Restrictions: We currently support only discrete 1D observations, continuous 1D observations, fixed-choices discrete actions, and variable-length discrete actions. Additional work will be required to expand our network generation code to handle other useful data types, such as vector-valued inputs and continuous actions.

Map Layout: Agents currently spawn at the edges of the map at a constant rate and forage inwards. This makes it difficult to study emergent cooperation in populations – it would be better if agents spawned in the same general area and could split off into groups. There are classic challenges associated with designing spawn areas in game development such as *spawn camping* – high level players sieging spawn and preventing new players from surviving long enough to have a fighting chance. We are working on a balanced central spawn implementation for the next platform update.

Randomization Limitations: We do not have randomization outside of the map level. It is possible to randomize more aspects of the game systems, and even learn them jointly with the policy. This is the great challenge of open-endedness and is a major target for future work.

Environment Resets: Neural MMO is persistent at test-time: we can run the environment for hours and watch agents over long time horizons. However, we are forced to periodically randomize the map at train-time. This limitation is entirely due to small available hardware scale. RLlib provides scalable parallelization – if we had 256 cores, we could simply train on one persistent map per core.

References

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. Emergent tool use from multi-agent autocurricula, 2019.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J. W., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. Dota 2 with large scale deep reinforcement learning. *ArXiv*, abs/1912.06680, 2019.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning, 2019.

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., et al. Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. *arXiv preprint arXiv:1807.01281*, 2018.

Liang, E., Liaw, R., Moritz, P., Nishihara, R., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I., and Stoica, I. Rllib: Abstractions for distributed reinforcement learning, 2017.

Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., and Stoica, I. Ray: A distributed framework for emerging ai applications. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI18, pp. 561577, USA, 2018. USENIX Association. ISBN 9781931971478.

OpenAI. Openai five. https://blog.openai.com/openai-five/, 2018.

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. Solving rubik's cube with a robot hand, 2019.

Perlin, K. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287296, July 1985. ISSN 0097-8930. doi: 10.1145/325165.325247. URL https://doi.org/10.1145/325165.325247.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W., Dudzik, A., Huang, A., Georgiev, P., Powell, R., Ewalds, T., Horgan, D., Kroiss, M., Danihelka, I., Agapiou, J., Oh, J., Dalibard, V., Choi, D., Sifre, L., Sulsky, Y., Vezhnevets, S., Molloy, J., Cai, T., Budden, D., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Pohlen, T., Yogatama, D., Cohen, J., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Apps, C., Kavukcuoglu, K., Hassabis, D., and Silver, D. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://deepmind.com/blog, 2019.