Doing More with More:

Recent Achievements in Large-Scale Deep Reinforcement Learning

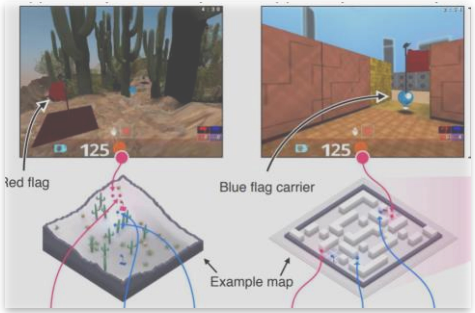
Compiled by: Adam Stooke, Pieter Abbeel (UC Berkeley)

March 2019

Atari, Go, and beyond

- Algorithms & Frameworks (Atari Legacy)
  - A3C / DQN (DeepMind)
  - IMPALA / Ape-X (DeepMind)
  - Accel RL (Berkeley)

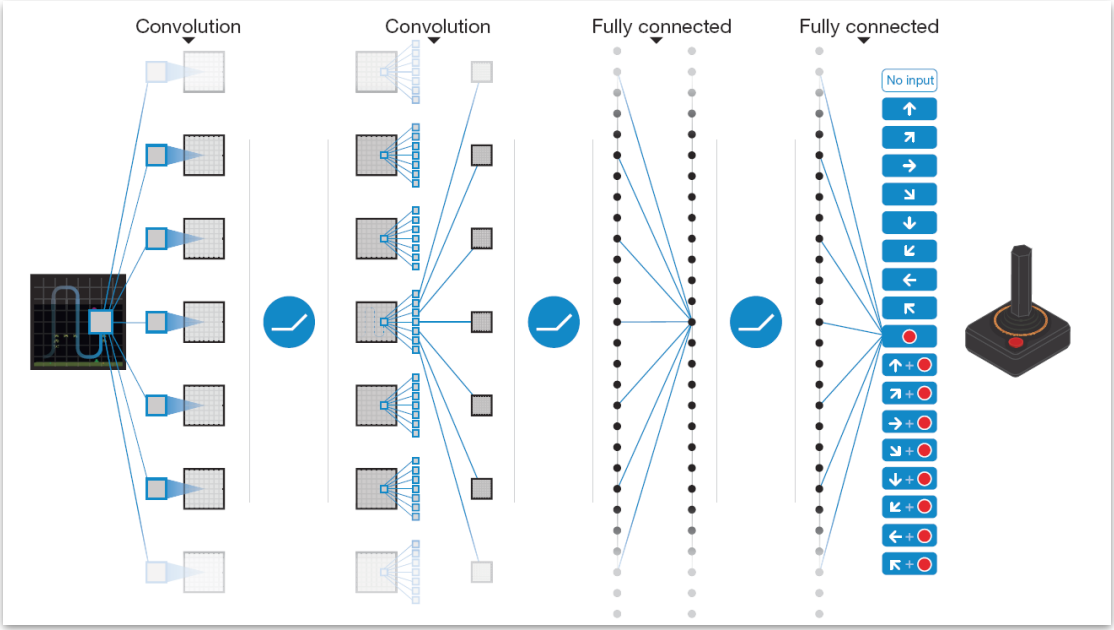
- Large-Scale Projects (Beyond Atari)
  - AlphaGo Zero (DeepMind)
  - Capture the Flag (DeepMind)
    - Population Based Training
  - Dota2 (OpenAI)
  - Summary of Techniques



Algorithms & Frameworks

(Atari Legacy)

“Classic” Deep RL for Atari



[Mnih, et al 2015]

Neural Network Architecture:

- 2 to 3 convolution layers
- Fully connected head
- 1 output for each action

“Classic” Deep RL for Atari

Deep Q-Learning (DQN): [Mnih, et al 2015]

- Algorithm:
  - Off-policy Q-learning from replay buffer
  - Advanced variants: prioritized replay, n-step returns, dueling NN, distributional, etc.
- System Config:
  - 1 actor CPU; 1 environment instance
  - 1 GPU training
- ~10 days to 200M Atari frames

Asynchronous Advantage Actor Critic (A3C): [Mnih, et al 2016]

- Algorithm:
  - policy-gradient (with value estimator)
  - asynchronous updates to central NN parameter store
- System Config:
  - 16 actor-learner threads running on CPU cores in one machine
  - 1 environment instance per thread
- ~16 hours to 200M Atari frames
  - (less intense NN training vs DQN)

SEAQUEST Training

top left Fully Random, Initial

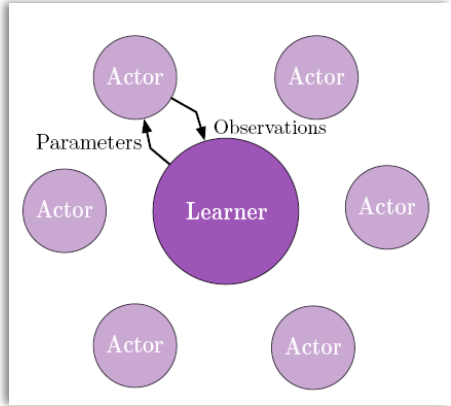
bottom left “Beginner” ~24M frames played

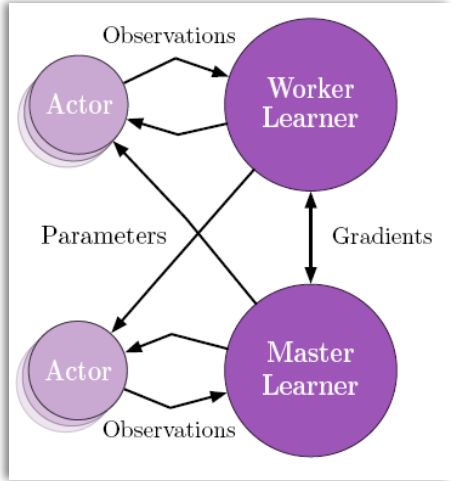
bottom right “Advanced” ~240M frames played

IMPALA

[Espeholt, et al 2018]

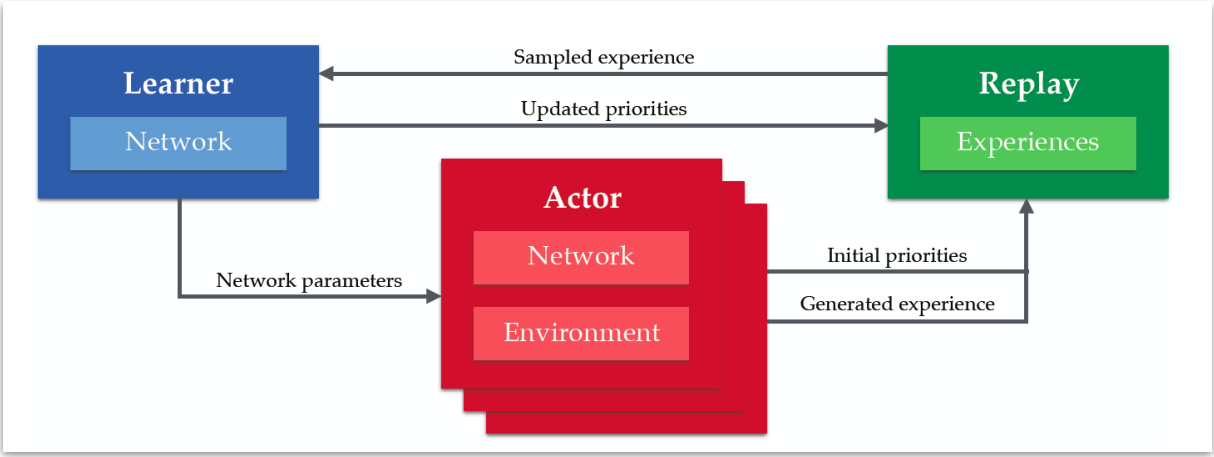
- System Config:
  - “Actors” run asynchronously on distributed CPU resources (cheap)
  - “Learner” runs on GPU; batched experiences received from actors
  - Actors periodically receive new parameters from learner
- Algorithm:
  - Policy gradient algorithm: descended from A3C
  - Policy lag mitigated through V-trace algorithm (“**Imp**ortance Weighted”)
- Scale:
  - Hundreds of actors, can use multi-GPU learner
  - (learned all 57 games simultaneously; speed not reported)





Ape-X

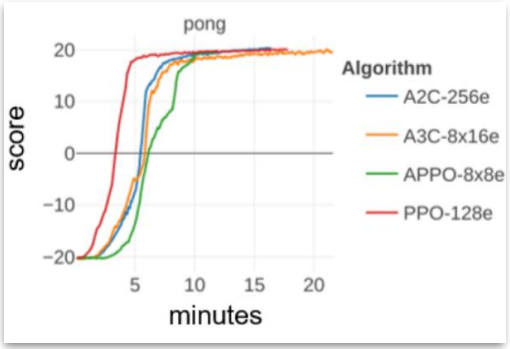
[Horgan, et al 2018]

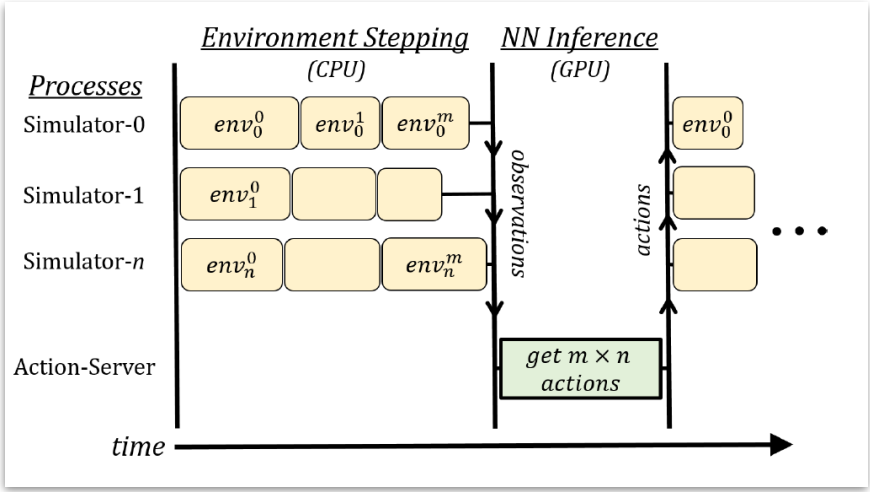


- Algorithm:
  - Off-policy, Q-Learning (e.g. DQN)
  - Replay buffer adapted for prioritization under distributed-actors setting
  - Hundreds of actors; using different ε in ε-greedy exploration improves scores
- System Config:
  - GPU learner, CPU actors (as in IMPALA)
  - Replay buffer may be on different machine from learner
- Scale:
  - 1 GPU, 376 CPU cores → 22B Atari frames in 5 days, high scores
  - (in large cluster, choose number CPU cores to generate data at rate of training consumption)

Accel RL

[Stooke & Abbeel 2018]





- System config:
  - GPU used for both action-selection and training -- batching for efficiency
  - CPUs each run multiple (independent) environment instances
  - CPUs step environments once, all observations gathered to GPU, GPU returns all actions, …
- Algorithms:
  - Both *policy gradient* and *Q-learning* algorithms
  - Synchronous (NCCL) and asynchronous multi-GPU variants shown to work
- Scale:
  - Atari on DGX-1: 200M frames ~1 hr; near linear scaling to 8 GPU, 40 CPU (A3C)
  - Effective when CPU and GPU *on same motherboard* (shared memory for fast communication)

Atari Scaling Recap

Algo/Framework	Compute Resources	Gameplay Generation Speed*	Training Speed**
DQN (original)	1 CPU; 1 GPU	230 frames per second	1.8K fps (8x generated)
Ape-X	376 CPU; 1x P100 GPU	50K fps	38.8K fps
Accel RL -- CatDQN	40 CPU; 8x P100 GPU	30K fps	240K fps (8x generated)
A3C (original)	16 CPU	3.5K fps	--
IMPALA	100's CPU; 8x GPU	?	?
Accel RL -- A2C	40 CPU; 8x P100 GPU	94K fps	--

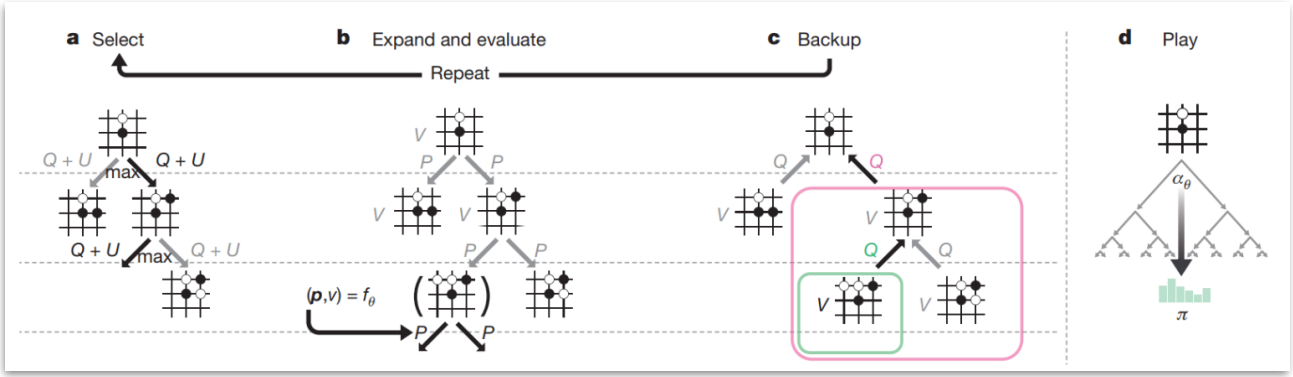
* i.e. algorithm wall-clock speed for learning curves
** 1 gradient per 4 frames; DQN standard uses each data point 8 times for gradients, A3C uses data once

Large-Scale Projects

(Beyond Atari)

AlphaGo Zero

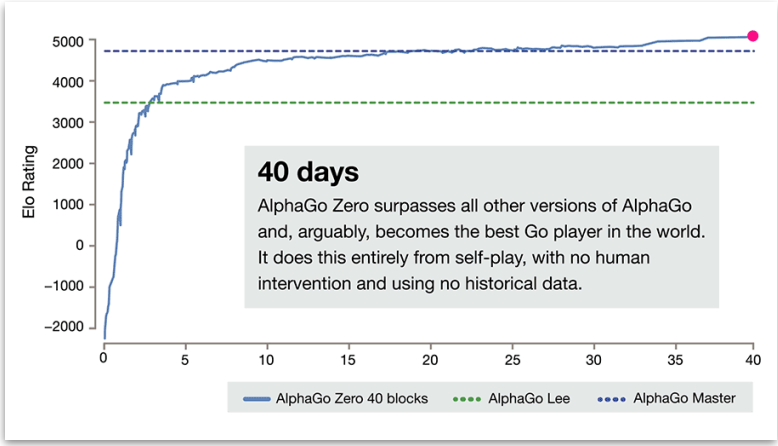
[Silver et al 2017]



- Algorithm:
  - Limited Monte-Carlo Tree Search (MCTS) guided by networks during play
    - After games, policy network trained to match move selected by MCTS
    - Value-estimator trained to predict eventual game winner
  - AlphaGo Fan/Lee (predecessors, 2015/2016):
    - Separate policy and value-prediction networks
    - Policy network initialized with supervised training on human play, before RL
  - AlphaGo Zero (2017):
    - Combined policy and value-prediction network, deeper
    - Simplified MCTS search
    - No human data: train with self-play and RL starting from fully random on raw board data

AlphaGo Zero

[DeepMind-AlphaGo-Zero-Blog]



- NN Architecture:
  - Up to ~80 convolution layers (in residual blocks)
  - Input: 19x19x17 binary values; last 7 board states

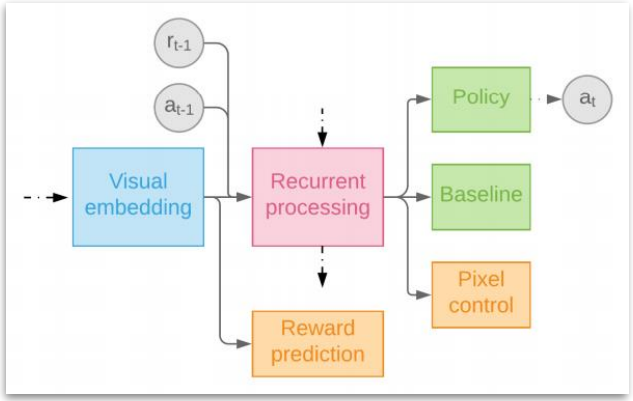
- Computational Resources:
  - Trained using 64 GPUs, 19 param server CPUs
    - Earlier versions of AlphaGo: 1,920 CPUs and 280 GPUs
  - MCTS: considerable quantity of NN forward passes (1,600 sims per game move)
  - Power consumption, decreasing by hardware and algorithm improvements:
    - AlphaGo Fan -- 176 GPUs: **40K TDP** (similar to Watts of electricity)
    - AlphaGo Lee -- 48 TPUs: **10K TDP**
    - AlphaGo Zero -- 4 TPUs: **1K TDP**

- Training Duration:
  - Final: 40 days training--29 million self-play games, 3.1 million gradient steps
  - By 3 days beat AlphaGo Lee

Capture the Flag [Jaderberg et al 2018]

- The Game:
  - First human-level performance in human-style, 3D first-person-action
  - 2v2 (multi-agent) game on custom maps on Quake III game engine

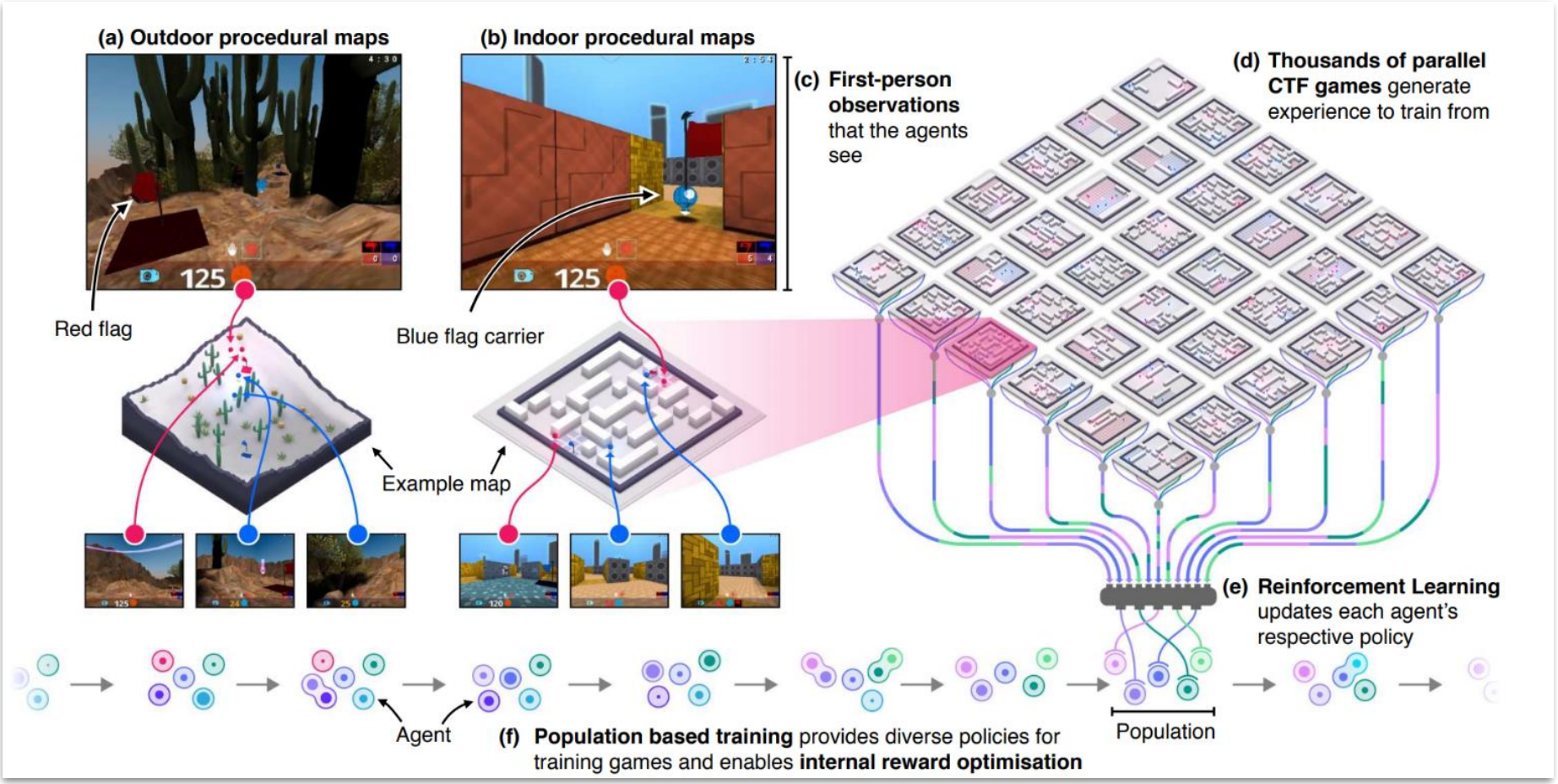
- NN Architecture:
  - 4 convolution layers (visual input: 84x84 RGB)
  - Differential Neural Computer (DNC) with memory
  - 2-level hierarchical agent (fast & slow recurrence)



- Algorithm:
  - IMPALA for training UNREAL agent (RL with auxiliary tasks for feature learning)
  - Population-based training, pop. size 30
  - Randomly assigned teams for self-play within population (matched by performance level)

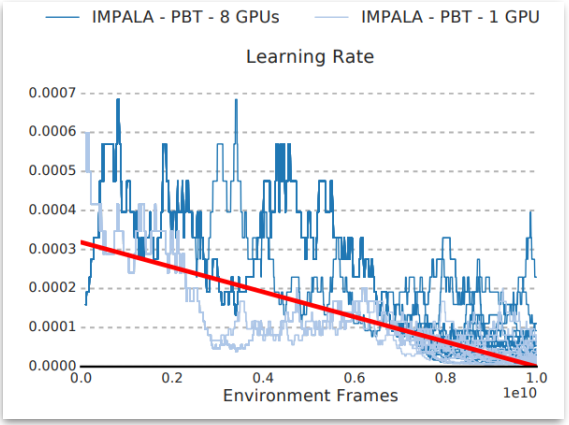
Capture the Flag

[Jaderberg et al 2018]



Population Based Training [Jaderberg et al 2017]

- Train multiple agents simultaneously and evolve hyperparameters.
  - Multiple learners; measure their relative performance
  - Periodically, poorly performing learner's NN parameters replaced from superior one
  - At same moment, hyperparameters (e.g. learning rate) copied and randomly perturbed
- More robust for achieving successful agent without human oversight / tuning
  - In CTF: evolved weighting of game events (e.g. picked up flag) to optimize RL reward
- Can discover schedules in hyperparameters
  - e.g. learning rate decay (vs red-line, hand-tuned linear decay)
- Use over any learning algorithm (e.g. IMPALA)
- Hardware/experiment scales with population size



First-person Agent Observation

Capture the Flag

- Computational Resources:
  - 30 GPUs for learners (1 per agent)
  - ~2,000 CPUs total for gameplay (sim & render--1000’s actors)
  - Experience fed asynchronously from actors to respective agent learner every 100 steps

- Training Duration:
  - Games: 5 minutes; 4,500 agent steps (15 steps per second)
  - Trained up to 2 billion steps, ~450K games (eq. 4 years gameplay, in roughly a week)
  - Beat strong human players by ~200K games played

Dota2

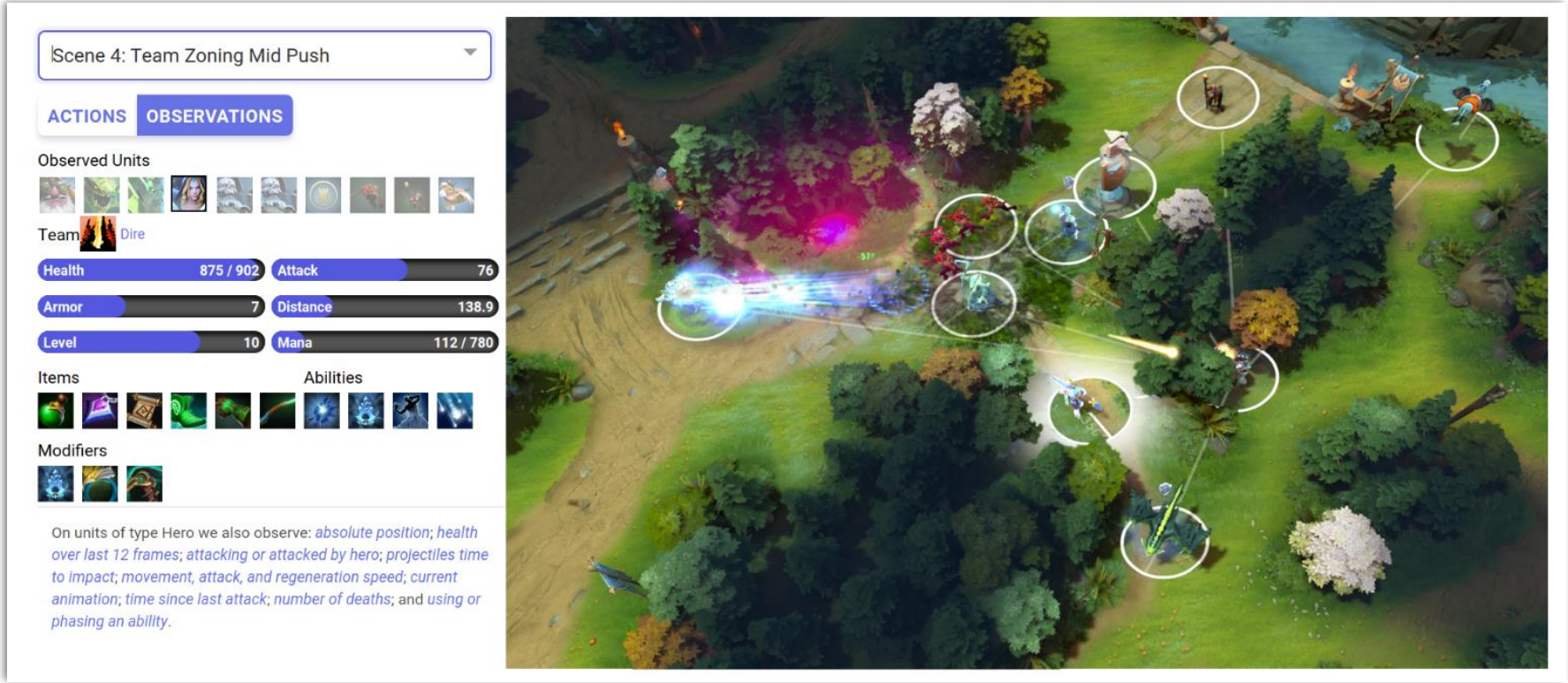
[OpenAI-Five-Blog]

- The Game:
  - Popular hero-based action-strategy
  - Massively scaled RL effort at OpenAI
  - Succeeded with 1v1 play
  - Now developing 5v5

- Algorithm:
  - PPO [Schulman et al 2017] (advanced policy gradient; multiple gradients per datum)
  - Trained by self-play from scratch
  - Synchronous updates across GPUs (all-reduce gradients using NCCL2)
  - Key to scaling: large training batch size for efficient multi-GPU use

Dota2

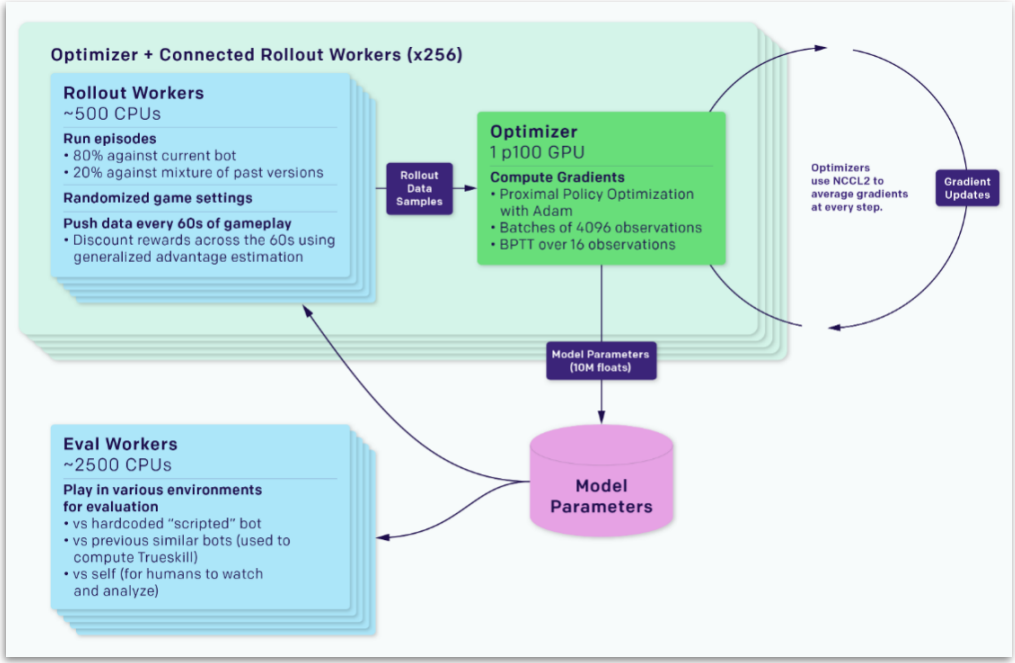
- NN Architecture:
  - Single-layer, 1,024-unit LSTM (10M params) // Separate LSTM for each player
  - Input: 20,000 numerical values (no vision) // Output: 8 numbers (170K possible actions)




Crystal Maiden's Freezing Field
and Viper's Nethertoxin force
the human team to scatter

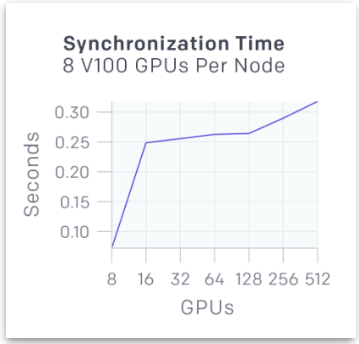
Dota2

[OpenAI-Five-Blog]



- Computational Resources:
  - 256 GPUs (P100), 128K CPU cores
  - ~500 CPUs rollouts per GPU
  - data uploaded to optimizer every 60s
  - (framework: “Rapid”)

- Training Duration:
  - Games: ~45 minutes; 20,000 agent steps (7.5 steps-per-second)
    - Go: ~150 moves per game
  - Train for weeks
  - 100’s years equivalent experience gathered per day



Large-Scale Techniques Recap

- 1000’s of parallel actors performing gameplay
  - (on relatively cheap CPUs)
- 10’s to 100’s GPUs for learner(s) (or ~10’s TPUs)
- Most daring examples so far using policy gradient algorithms, not Q-learning
  - Asynchronous data transfers → learning algorithm must handle slightly off-policy data
- Billions of samples per learning run to push the limits in complex games
- Self-play pervasive, in various forms
- Research efforts require significant multiples of listed compute resources
  - Development requires experimentation with many such learning runs

References

1. **A3C**: Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." International conference on machine learning. 2016. https://arxiv.org/abs/1602.01783
2. **DQN**: Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529. https://arxiv.org/abs/1312.5602
3. **IMPALA**: Espeholt, Lasse, et al. "IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures." arXiv preprint arXiv:1802.01561 (2018). https://arxiv.org/abs/1802.01561
4. **APE-X**: Horgan, Dan, et al. "Distributed prioritized experience replay." arXiv preprint arXiv:1803.00933 (2018). https://arxiv.org/abs/1803.00933
5. **Accel RL**: Stooke, Adam, and Pieter Abbeel. "Accelerated methods for deep reinforcement learning." arXiv preprint arXiv:1803.02811 (2018). https://arxiv.org/abs/1803.02811
6. **PBT**: Jaderberg, Max, et al. "Population based training of neural networks." arXiv preprint arXiv:1711.09846 (2017). https://arxiv.org/abs/1711.09846
7. **AlphaGo**: Silver, David, et al. "Mastering the game of Go without human knowledge." Nature 550.7676 (2017): 354. Nature paper
8. **AlphaGo Zero Blog**: https://deepmind.com/blog/alphago-zero-learning-scratch/
9. **CTF**: Jaderberg, Max, et al. "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning." arXiv preprint arXiv:1807.01281 (2018). https://arxiv.org/abs/1807.01281
10. **Dota2 Blog**: https://blog.openai.com/openai-five/
11. **PPO**: Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017). https://arxiv.org/abs/1707.06347