[FORTECH.AI]

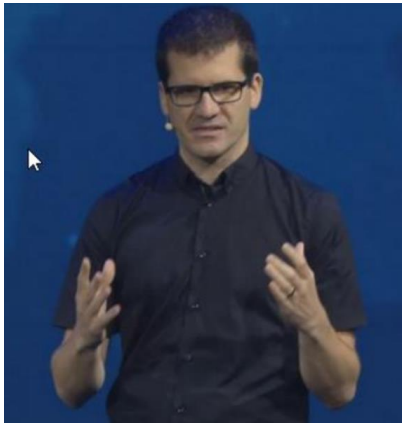
[IASI AI]

Alphastar DRL Agent

Alex Movila

[IASI AI] Alphastar AI - Team Lead

Oriol Vinyals:
- Part of Google Brain before
- His research is used in Google Translate, Text-To-Speech and Speech recognition
- Cited over 43000 times



David Silver:

Professor of Computer Science of University College London
- Lead researcher of AlphaGo/AlphaZero
- Cited over 29000 times

[IASI AI] StarCraft II: What and Why

- Real-time strategy game: gather resources, build technology, defeat opponent
- Complexity: among video games, considered to be at the peak of human ability
- Canonical: played by millions, esport endured 20 years of active human play
- Research : hundreds of submissions over 12 years of competition

“It is kind of like chess on steroids”

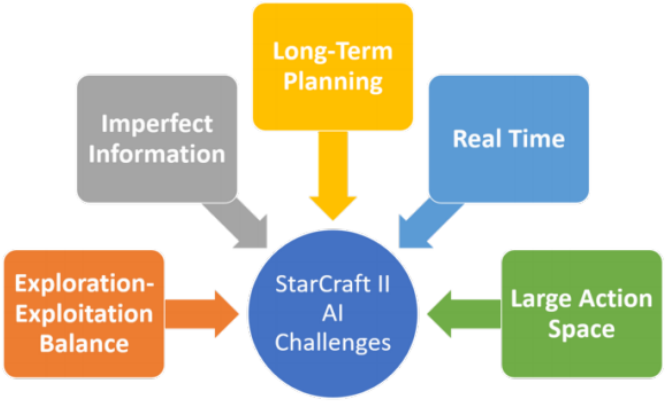
“The skill ceiling is unreachable due to mechanical limits”

“StarCraft is impossible to play perfect”

[IASI AI] Why Starcraft?

Balance the short and long-term goals and adapt to unexpected situations requires:

- **Real time:** continually perform actions for each player
- **Imperfect information:** Only see information in the camera view. Requires ”actively scouting”
- **Long term planning:** Early actions may payoff later. Credit assignment problems.
- **Large action space:** a myriad of units and buildings to be controlled simultaneously; a mass of combinatorial and hierarchical space for decision-making.
- **Game theory:** There is no single superior strategy (rock-paper-scissors). Counter-strategies discovered by pro players over 20 years. Continual exploration and exploitation



Go:
- Perfect information
- 361 action space
- 1 player combination
- Simple environment

StarCraft:
- Imperfect information
- 1026 action space
- 6 player combinations
- Complex environment

[IASI AI] What has happened? – A new star is born

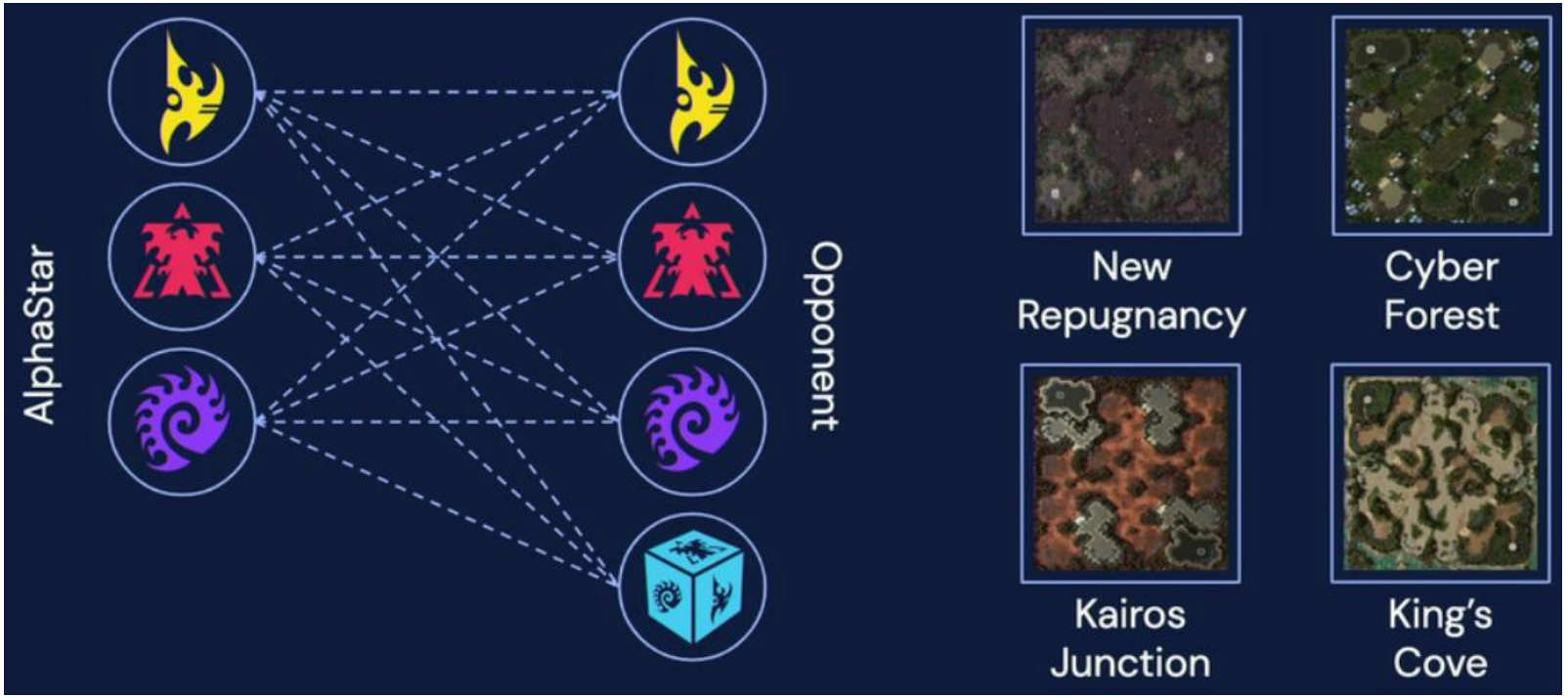
Ranked above 99.8% of human players

- December 10th 2018: AlphaStar beats the best DeepMind Starcraft player
- December 12th 2018: AlphaStar beats Dario “TLO” Wünsch, a Pro Starcraft Player
  - BUT: TLO plays Zerg normally
- December 19th 2018: AlphaStar beats Grzegorz “MaNa” Komincz, a Pro Starcraft Protoss Player

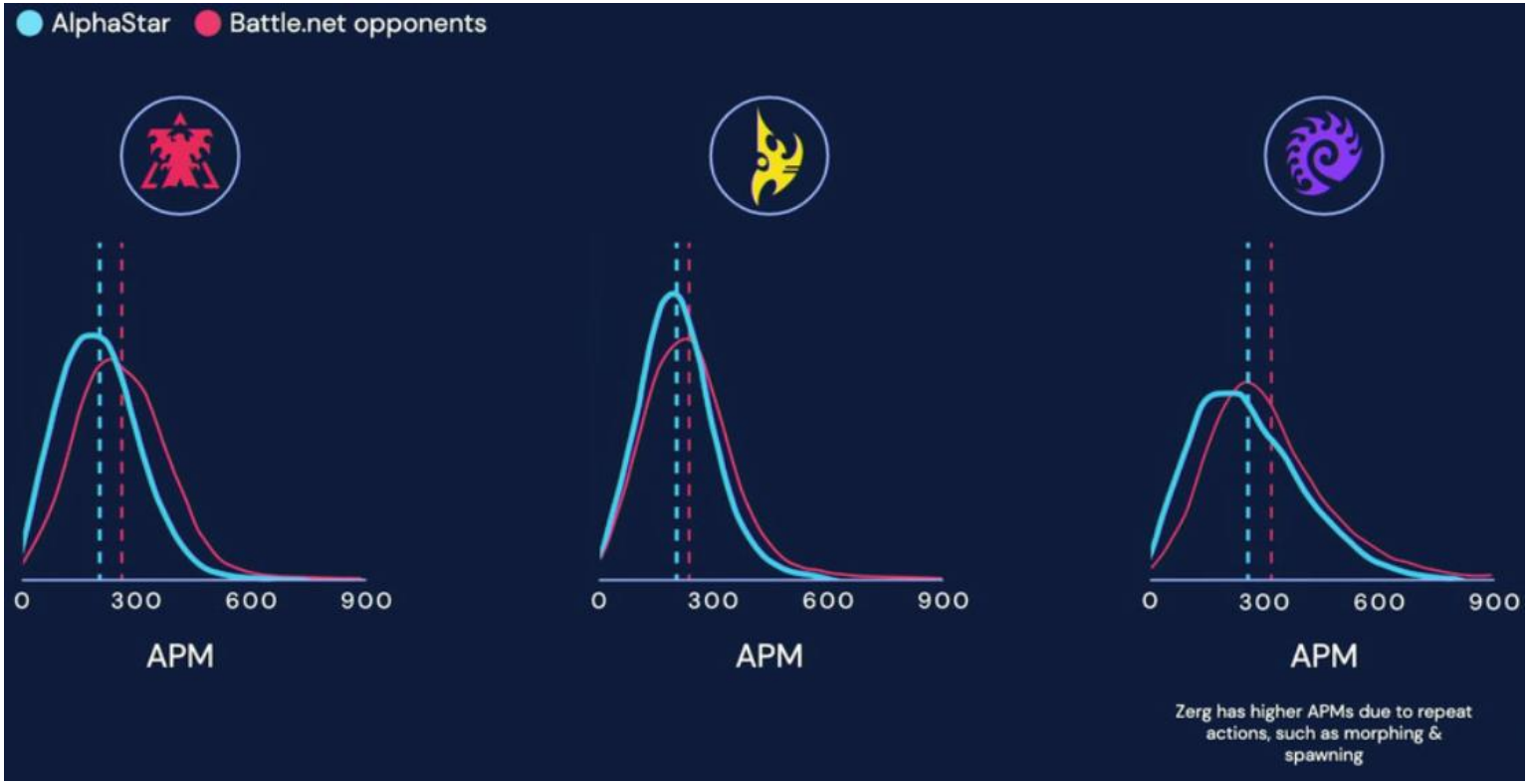
AlphaStar v1 (was cheating a bit....)

- Demoed in late 2018
- Complete visibility of entire map
- Capable of superhuman spikes in action rate
- Always played with/against the Protoss race

[IASI AI] Now, all races & more maps



Reduced APM - Pro approved



Limit avg APM to be human and reduce superhuman APM spikes of previous version

[IASI AI] Uses camera view - Pro approved

In Jan it used zoom out camera view so it saw entire map at once, now updated to move camera like a human

Pros comments about Alphastar

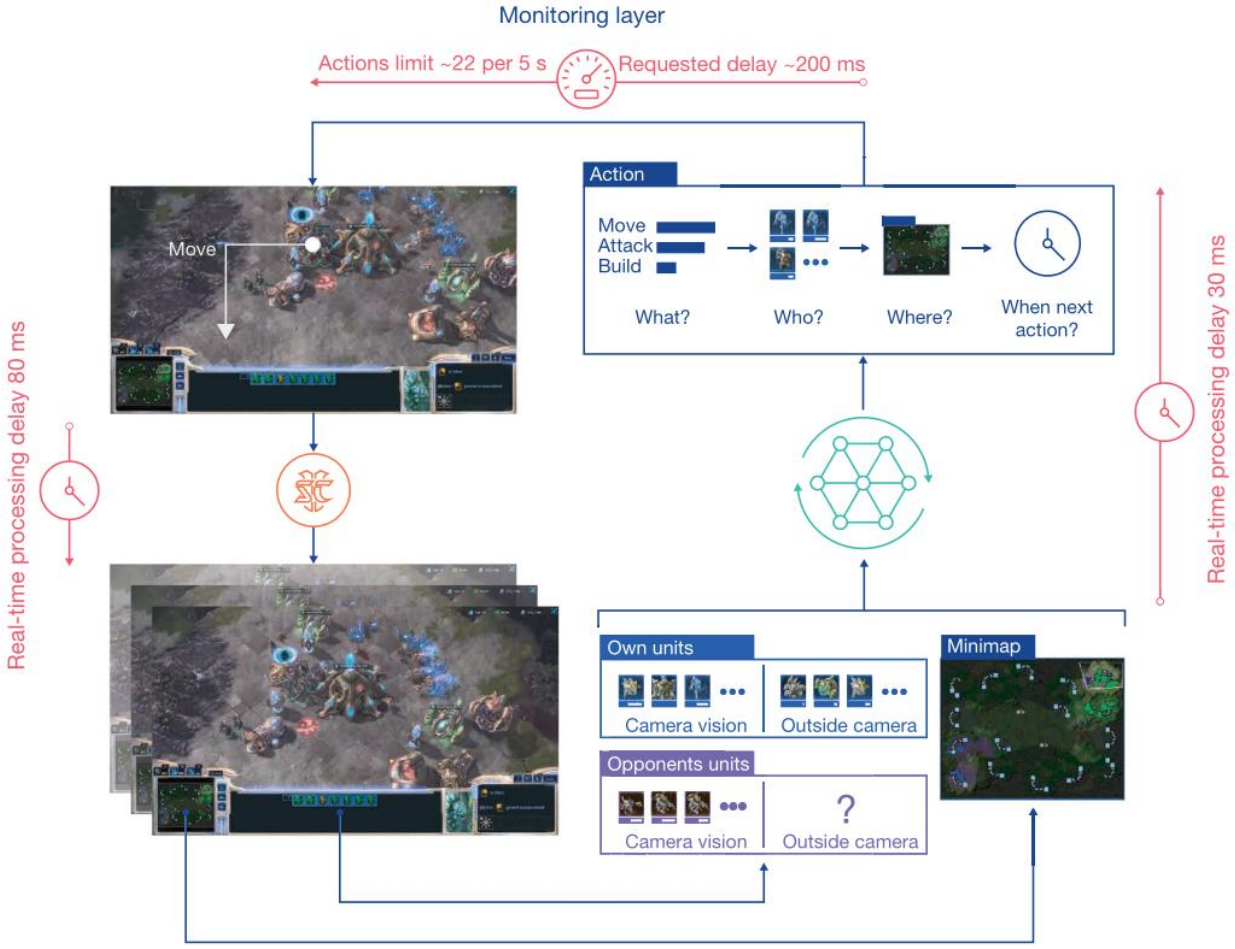
AlphaStar vs Serral - An Introduction (Alphastar won 4:1 at BlizzCon (nov 19))

How does AlphaStar work?

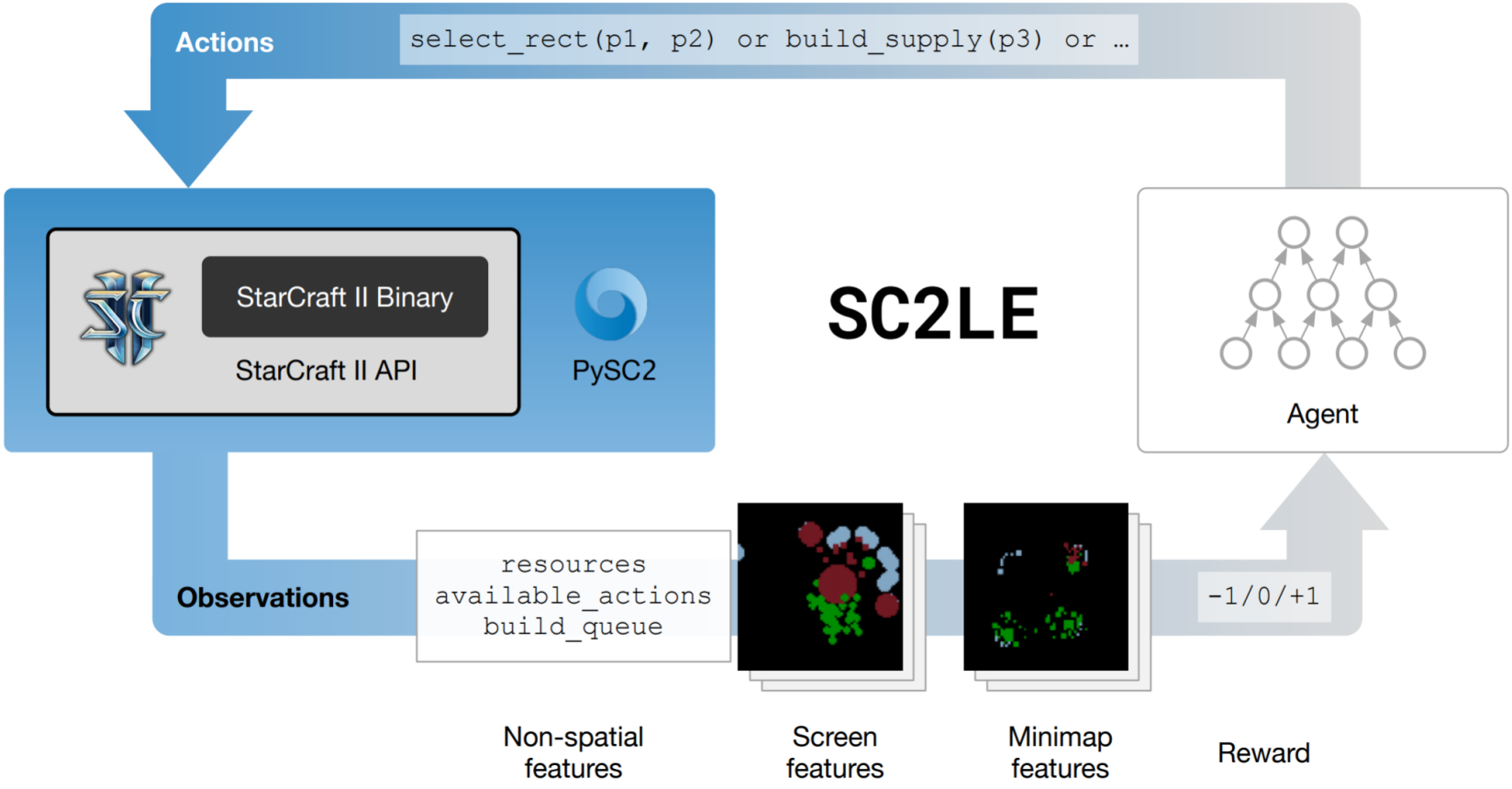
1. At step t, the agent receives an observation Ot (imperfect).

2. For each action At with aprox 10^26 possible choices:
- what action type
- who to issue that action to
- where to target
- when to observe and act next

3. Limit reaction time and action rates (APM 22 per 5 secs)



SC2LE – Starcraft 2 Learning Environment



StarCraft II: A New Challenge for Reinforcement Learning , Github

Scale and Compute

1. ATARI

   200M frames; 1000 hours / game

2. CTF

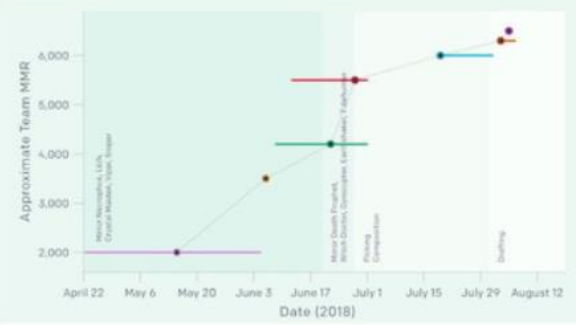
   450K games / 4 game years / agent

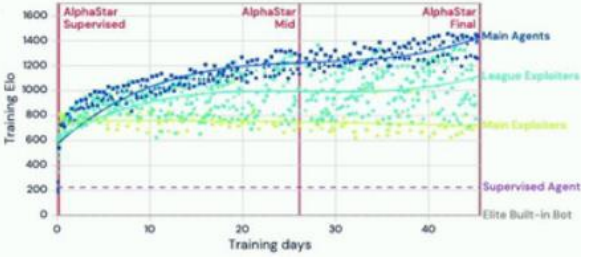
3. OpenAI Five

   45000 game years

4. AlphaStar

   200 game years / agent





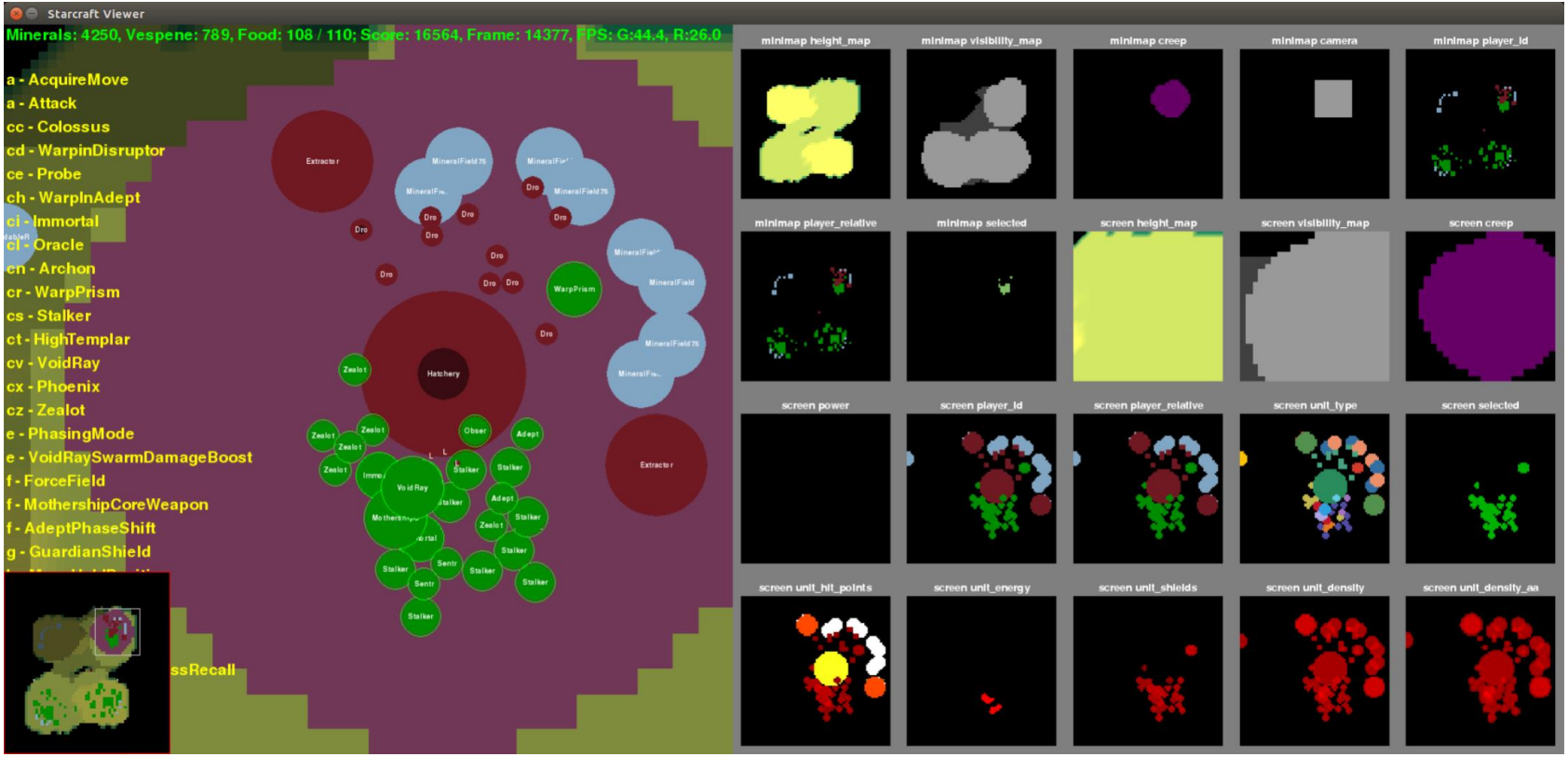
[IASI AI] Observations

Use feature layers instead of 3D image

- Main map
- Minimap
- Interface

AlphaStar: Mastering the Real-Time Strategy Game Starcraft II

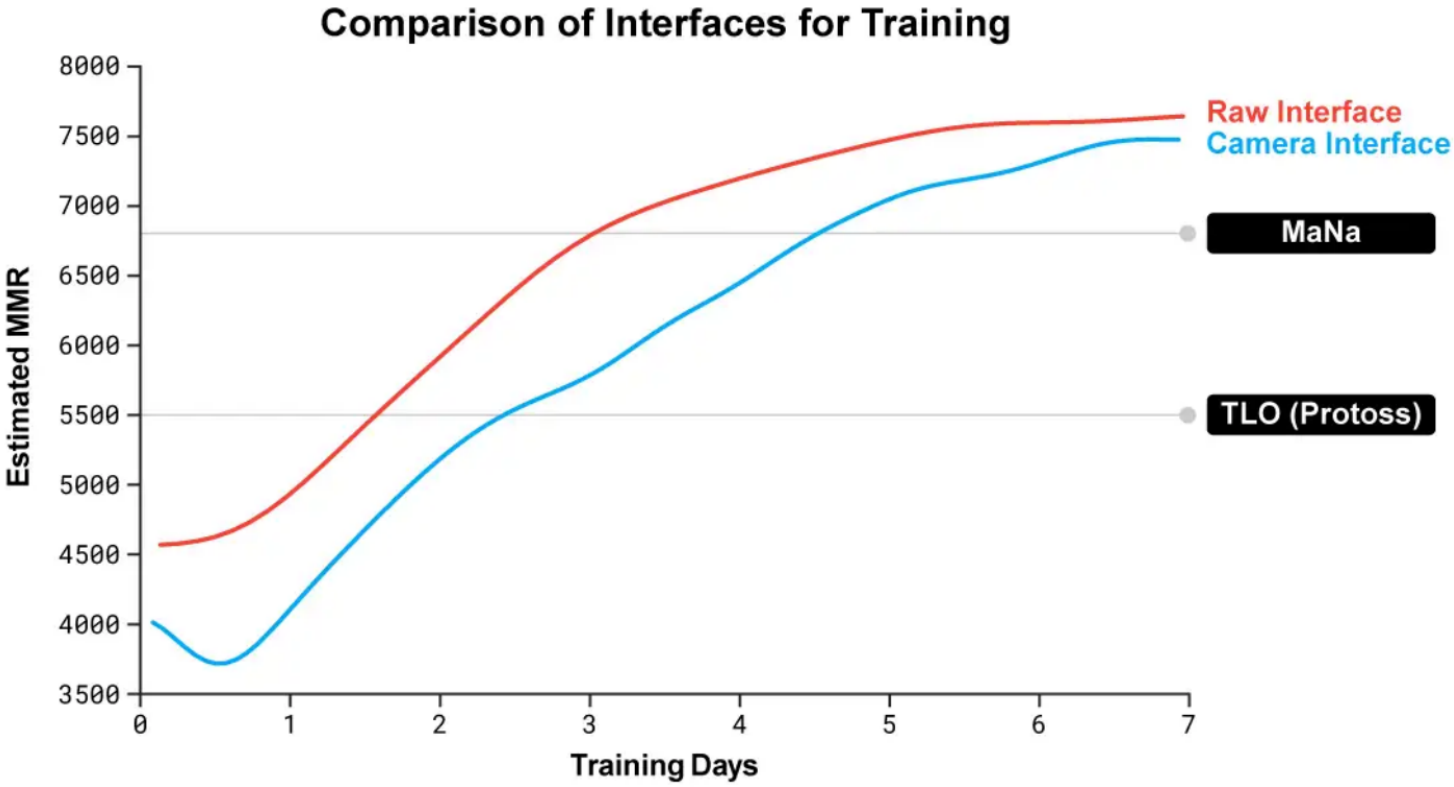
[IASI AI] Observations



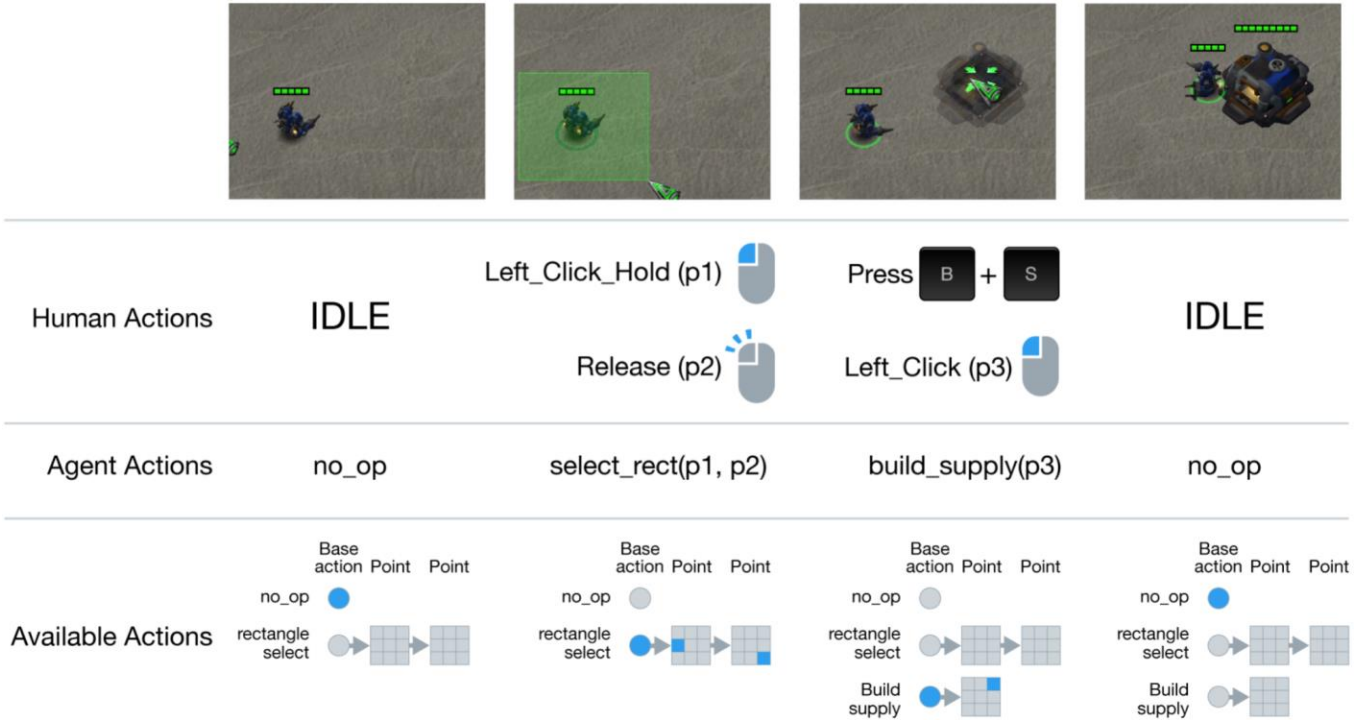
The PySC2 viewer shows a human interpretable view of the game on the left, and coloured versions of the feature layers on the right. For example, terrain height, fog-of-war, creep, camera location, and player identity, are shown in the top row of feature layers. (Youtube)

[IASI AI] Feature engineering also matters!

Another NN trained with camera-based interface lost the follow-up game against MaNa.



[IASI AI] Actions



Comparison between how humans act on StarCraft II and the actions exposed by PySC2. We designed the action space to be as close as possible to human actions. The first row shows the game screen, the second row the human actions, the third row the logical action taken in PySC2, and the fourth row the actions a exposed by the environment (and, in red, what the agent selected at each time step). Note that the first two columns do not feature the 'build supply' action, as it is not yet available to the agent in those situations as a worker has to be selected first.

[IASI AI] Input features

Category	Field	Description
Entities: up to 512	Unit type	E.g. Drone or Forcefield
	Owner	Agent, opponent, or neutral
	Status	Current health, shields, energy
	Display type	E.g. Snapshot, for opponent buildings in the fog of war
	Position	Entity position
	Number of workers	For resource collecting base buildings
	Cooldowns	Attack cooldown
	Attributes	Invisible, powered, hallucination, active, in cargo, and/or on the screen
	Unit attributes	E.g. Biological or Armored
	Cargo status	Current and maximum amount of cargo space
	Building status	Build progress, build queue, and add-on type
	Resource status	Remaining resource contents
	Order status	Order queue and order progress
	Buff status	Buffs and buff durations
Map: 128x128 grid	Height	Heights of map locations
	Visibility	Whether map locations are currently visible
	Creep	Whether there is creep at a specific location
	Entity owners	Which player owns entities
	Alerts	Whether units are under attack
	Pathable	Which areas can be navigated over
	Buildable	Which areas can be built on
Player data	Race	Agent and opponent requested race, and agent actual race
	Upgrades	Agent upgrades and opponent upgrades, if they would be known to humans
	Agent statistics	Agent current resources, supply, army supply, worker supply, maximum supply, number of idle workers, number of Warp Gates, and number of Larva
Game statistics	Camera	Current camera position. The camera is a 32x20 game-unit sized rectangle
	Time	Current time in game

Action Properties

Field	Description
Action type	Which action to execute. Some examples of actions are moving a unit, training a unit from a building, moving the camera, or no-op. See PySC2 for a full list[7]
Selected units	Entities that will execute the action
Target	An entity or location in the map discretised to 256x256 targeted by the action
Queued	Whether to queue this action or execute it immediately
Repeat	Whether or not to issue this action multiple times
Delay	The number of game time-steps to wait until receiving the next observation

[IASI AI] Exploration Challenge

- ❑ Exploration is very challenging
- Large action-space
- State-dependent actions
- Agents learn naive strategies

E.g. using just self-play => rush all workers to atack enemy camp

- ❑ Many mechanics have vanishingly small probability for a random policy

E.g. Expanding <10^-8

Self-play starting from random seems to difficult – local optimum

Solutions:

- Pseudo-rewards – require tweaking
- Imitation Learning

Walling:

"Even with a strong self-play system, there would be almost no chance of a system developing successful strategies in such a complex environment without some prior knowledge."

[IASI AI] Thoughts of diversity and exploration

- Naive exploration in micro-tactics could lead to a huge waste of computation. AlphaStar adopts z statistics to maintain the diversity.

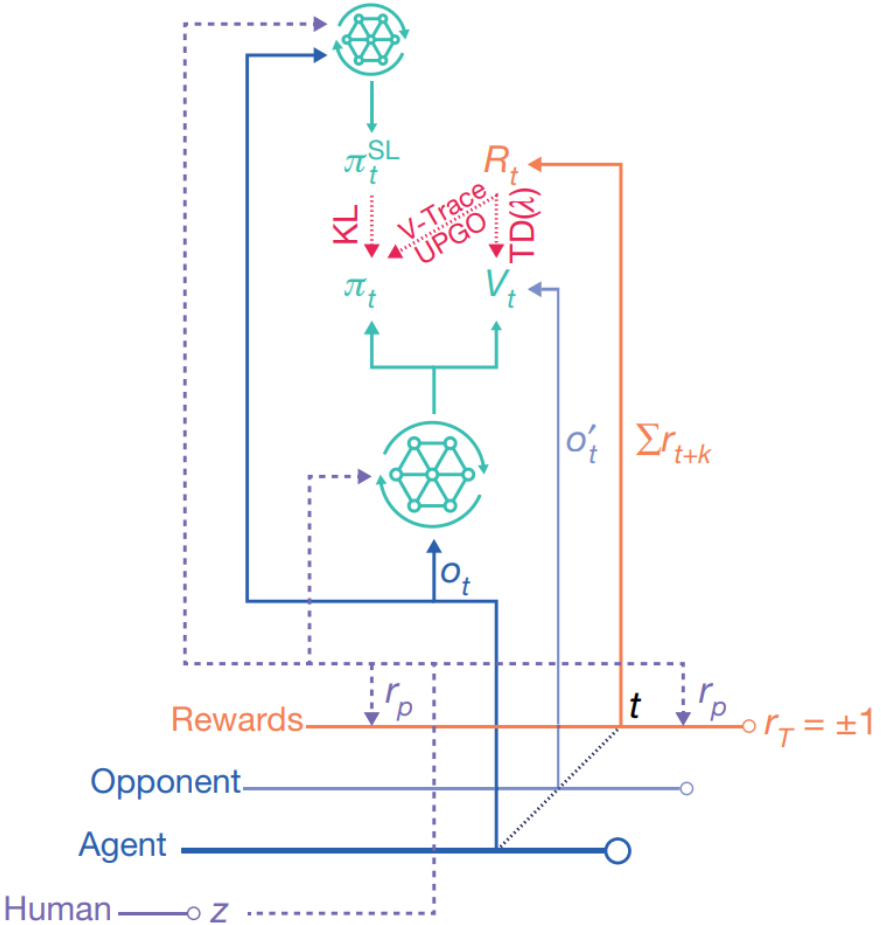
Z statistics encodes:
- **player build order**, defined as the first 20 constructed buildings and units
- **cumulative statistics**, defined as the units, buildings, effects, and upgrades that were present during a game

- To avoid naive exploration, use pseudo-rewards measure the edit distance between sampled and executed build orders, and the Hamming distance between sampled and executed cumulative statistics

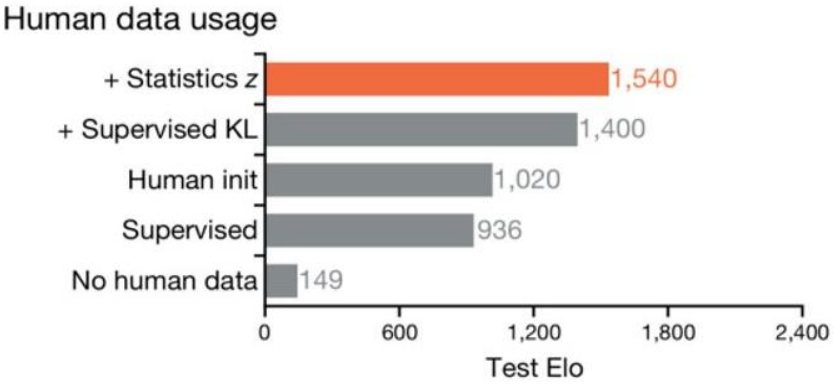
```
1  def hamming_distance(s1, s2):
2          """ Return the Hamming distance between equal-length sequences """
3      if len(s1)!=len(s2): raise ValueError("Non-equal length given!")
4      return sum(e1!=e2 for e1,e2 in zip(s1,s2))
```

- Grounding on this, add constraints like $\mathbb{KL}(\theta|\theta_{SL})$.

[IASI AI] Self-play RL – not enough



$$\pi_0 = \pi^{SL}$$
$$KL(\pi, \pi^{SL})$$
$$z \sim p(z)\ \pi(.|z)$$



Metric: ELO against a held-out set of agents

Self-play = 149
Supervised (no RL) = 936
Human init = Supervised + RL with only reward is win/lose = colapse to a single strategy
+ Supervised KL = add distillation loss (KL between RL policy and human prior)

Neurips talk

[IASI AI] Imitation Learning – stay close to human actions and its z stats

- 971,000 human replays from the top 22% of players
- For each player the statistic ‘z’ characterizes actions

Behaviour cloning of human players

Metric: ELO against a held-out set of agents.

Summarize stats of human players + build order in z statistics

Z Statistics:
A player’s build order and cumulative statistics

- First 20 constructed buildings and units
- Units, buildings, effects, upgrades present during a game
- Set to zero 10% of the time

Neurips talk

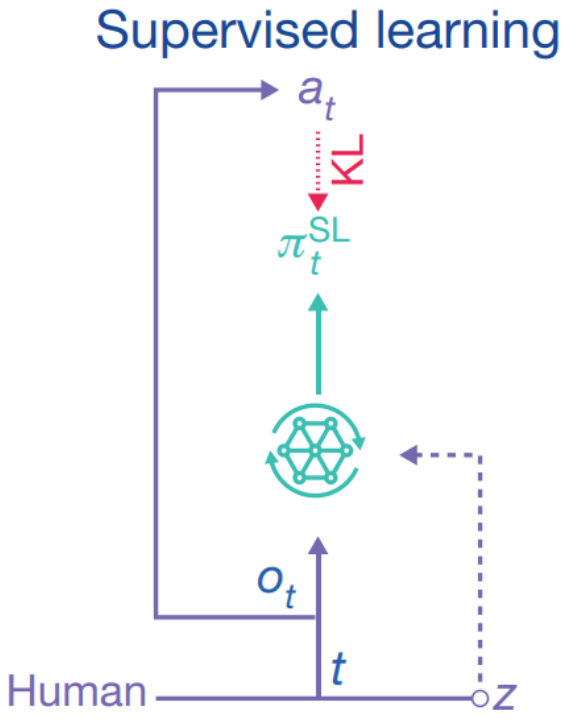
Imitation Learning with Supervised training

Supervised Learning – Training

- At each step input the current observations
- Output a probability distribution over each action
- Compute KL divergence between policy and z
- KL divergence between human actions and the policy's outputs
- Compute L2 regularization
- Apply updates using Adam optimizer

Supervised Learning – Fine-tuning

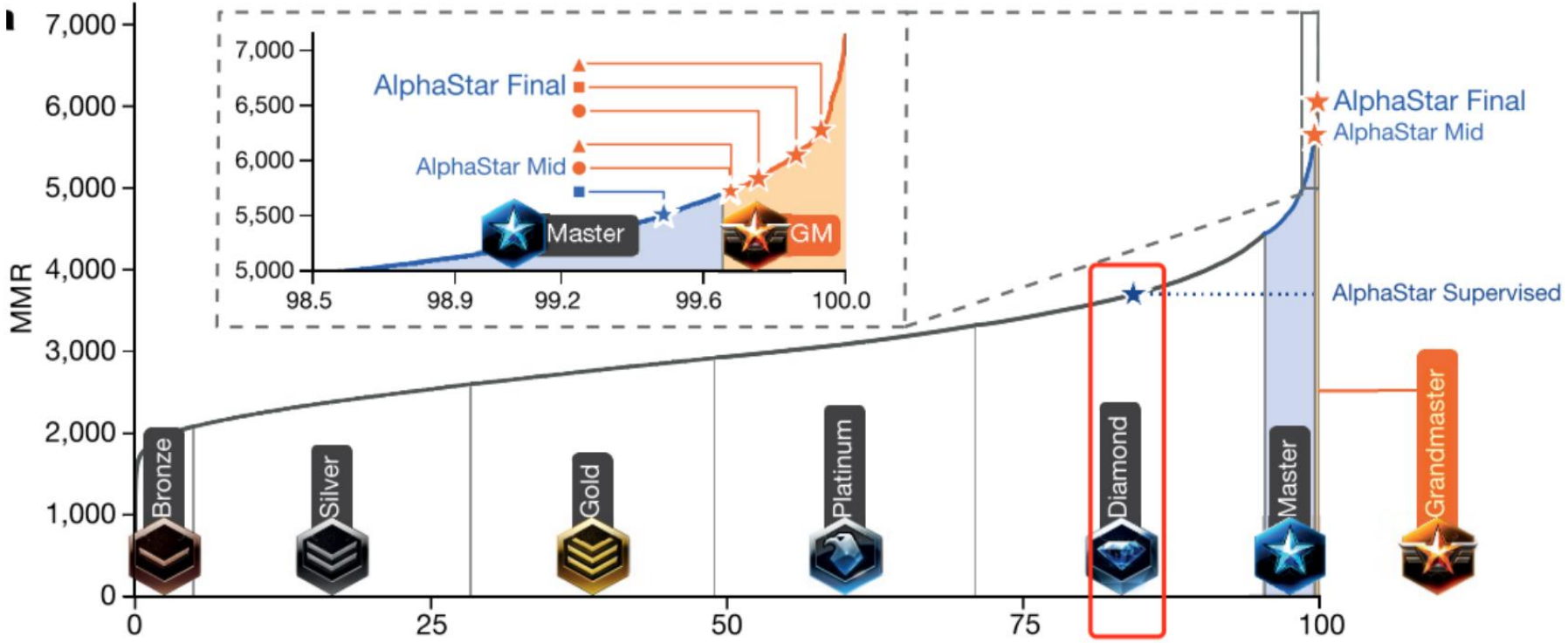
- Fine-tuned the policy on 16,000 games from top players
- Agents are capable of building all units in the game
- Qualitatively diverse from game to game



Neurips talk

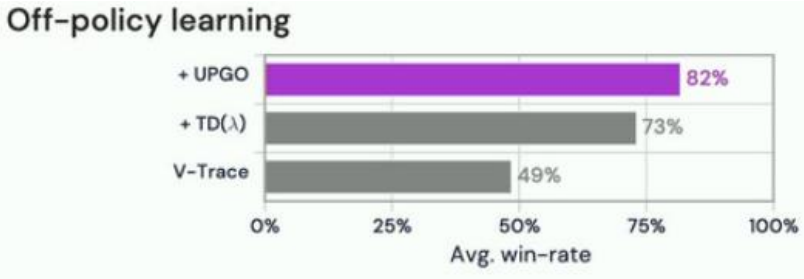
[IASI AI] Contributions of SL and RL

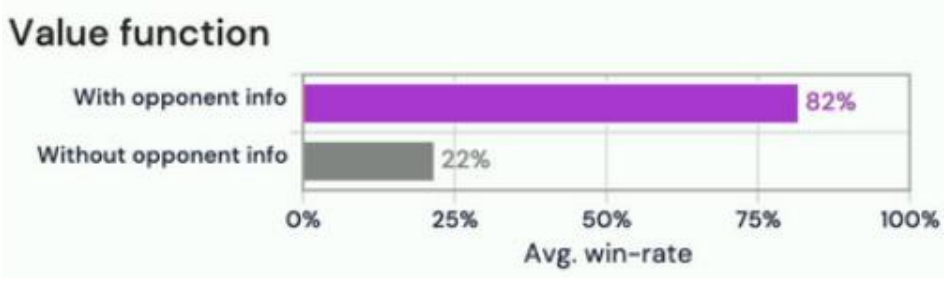
Pure supervised training could put the agent at the Diamond level!
AlphaStar Supervised reached an average rating of 3,699, which places it above 84% of human players and shows the effectiveness of supervised learning.



[IASI AI] Value Update

- The quality of the value function is critical.
- Existing off-policy correction methods were not efficient in SC2.
  - Many different action sequences can lead to (almost) identical behaviours.
  - Vanilla **TD(λ)** was better than V-trace. [Espeholt & al, 2018]
  - **UPGO**, an improved version of self-imitation learning, performed best
- Giving opponent information to value function helps [Kraemer & Banerjee, 2016]





Metric: Average win-rate against a fixed training set of agents

Neurips talk

Policy, State, Value

a value function is trained to predict Rt:

$$V_\theta(s_t, z) \qquad s_t = (o_{1:t}, a_{1:t-1})$$

and used to update the policy:

$$\pi_\theta(a_t \mid s_t, z) = \mathbb{P}[a_t \mid s_t, z]$$

[IASI AI] Reinforcement Learning algorithms

Reward: match outcome without discount
  - Win = 1, Tie = 0, Loss = -1
- Advantage **Actor-critic** paradigm used
  - Value function predicts rewards and updates policy
- Pseudo rewards for following **“z” human player strategy statistics**
- ‘Actors’ generate state-action trajectories
- ‘Learners’ asynchronously update model parameters
  - Using a **replay buffer** to store trajectories

Off-policy learning =>
In large action spaces, the current and previous policies are highly unlikely to match over many steps =>
use a combination of techniques that can learn effectively despite the mismatch:

Policy:
- Policy updated using **V-trace (clipped importance sampling)** for off-policy corrections for the policy (which avoids instability)
- and upgoing policy update **UPGO** for self-imitation learning that moves the policy towards trajectories with better-than-average reward

Value:
- Value estimates are updated using **vanilla TD(λ) (temporal difference learning) for** an uncorrected update of the value function (which introduces bias but reduces variance)
  - uses information from both the player's and the opponent's perspectives

Reinforcement Learning — TD(λ) Introduction
Combining TD and MC Learning

[IASI AI] How does it train?

Supervised learning (main contribution!)

- AlphaStar is initially trained with **supervised learning** (SL) with anonymous game replays from human experts
- This offers a good initialization for neural networks
- This initial agent beat the built-in "Elite" level AI ($\approx$ human golden level)

Reinforcement learning

"We found our use of human data to be critical in achieving good performance with reinforcement learning"

- $\theta_0 \leftarrow \theta_{\mathrm{SL}}$
- Model $\pi_\theta(a_t|s_t, z) = \mathbb{P}[a_t|s_t, z]$, where z summarizes the strategies sampled from a human data
- Seed a multi-agent RL with a continuous league
- Adopt *population-based training* (PBT) and *multi-agent* RL
- The league training can be treated as a bootstrapping DAgger process

[IASI AI] Alphastar AI – Training

- It's trained with IMPALA (Espeholt et al, 2018). (distributed training – multi actor – multi learner)
- It also uses experience replay. (replay buffer)
- And self-imitation learning (Oh et al, ICML 2018). (this is new UPGO which is a better variant)
- And policy distillation in some way (Rusu et al, ICLR 2016).
  (apply model compression using distillation specially adapted for RL)
- Which is trained with population-based training (Jaderberg et al, 2018).
- And a reference to Game-Theoretic Approaches to Multiagent RL (Lanctot et al, NeurIPS 2017). I'm not sure where this is used. Possibly for adding new agents to the AlphaStar league that are tuned to learn the best response to existing agents?

An Overdue Post on AlphaStar

[IASI AI] Alphastar AI – Impala setup

Actors send sequences of observations, actions, and rewards over the network to a central 128-core TPU learner worker, which updates the parameters of the training agent. The received data are buffered in memory and replayed twice. The learner worker performs large-batch synchronous updates. …

The learner processes about 50,000 agent steps per second. The actors update their copy of the parameters from the learner every 10 s.

We instantiate 12 separate copies of this actor–learner setup: one main agent, one main exploiter and two league exploiter agents for each StarCraft race. One central coordinator maintains an estimate of the payoff matrix, samples new matches on request, and resets main and league exploiters. Additional evaluator workers (running on the CPU) are used to supplement the payoff estimates.

[IASI AI] Alphastar – Training infrastructure

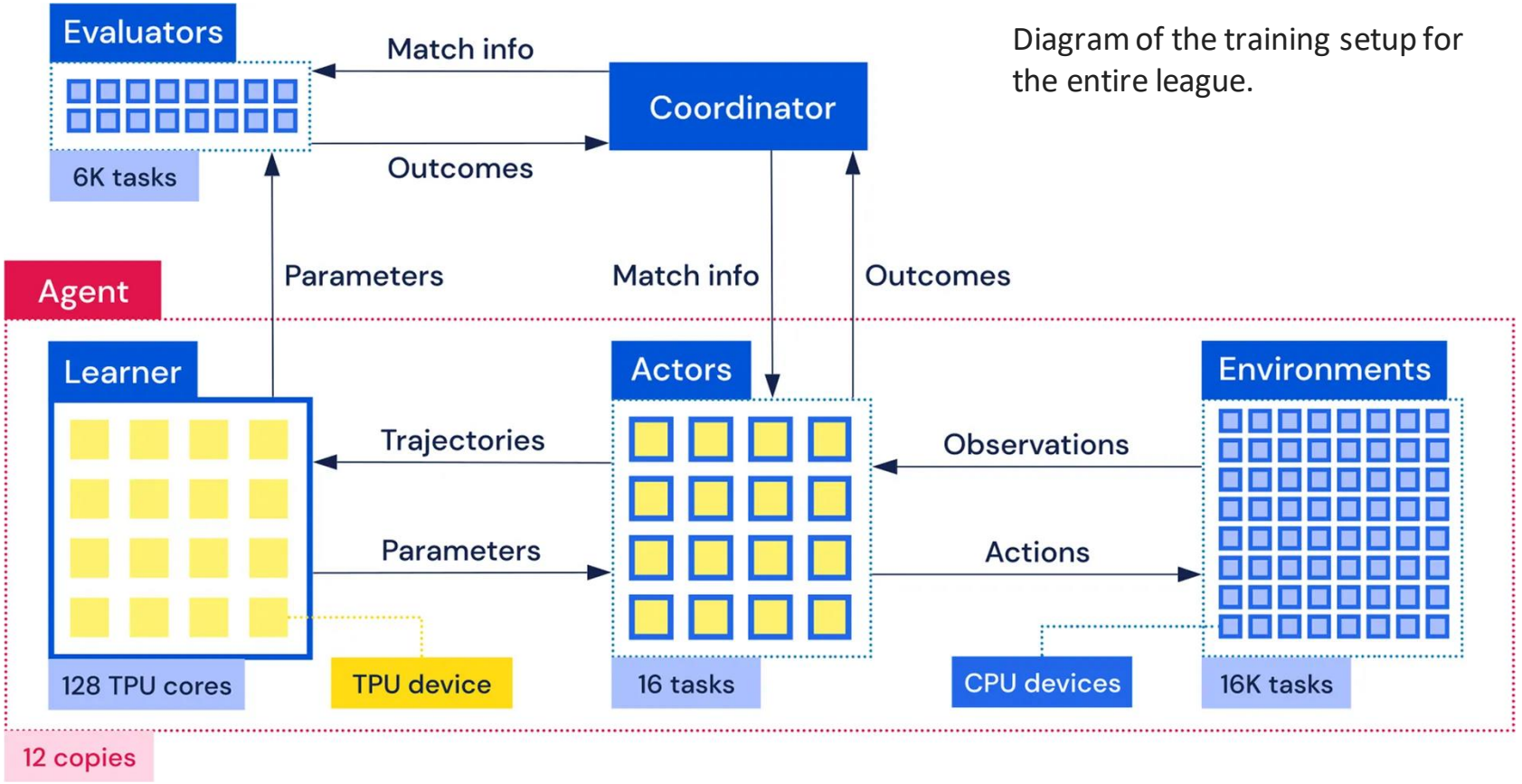


Diagram of the training setup for the entire league.

Extended Data Fig. 6: Training infrastructure

[IASI AI] Alphastar Model – Toss things together

AlphaStar aggregates a couple of SOTA approaches in terms of the *long-term sequence modeling* and *large output dimension*, e.g. machine translation, language modeling, visual representations.

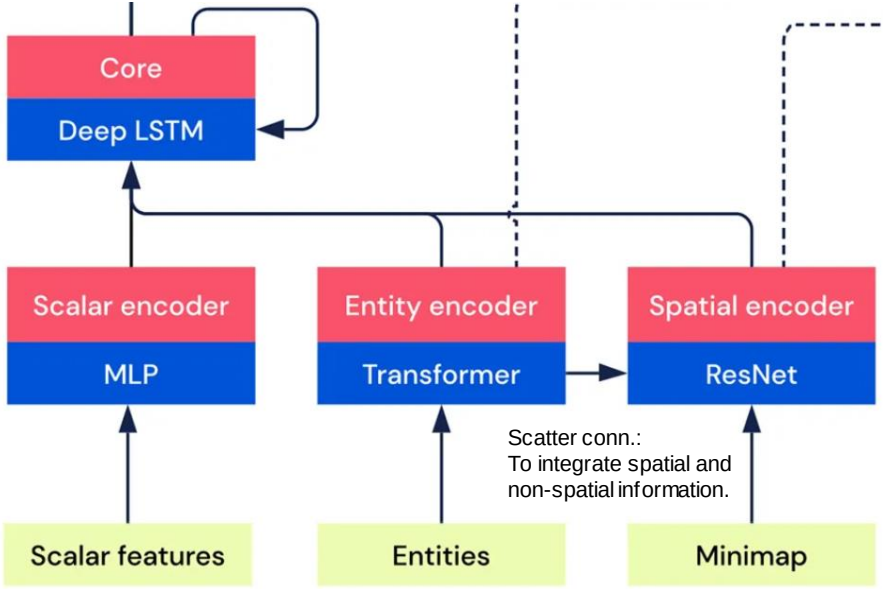
- ❑ Transformer
- ❑ Relational inductive biases
- ❑ LSTM core
- ❑ Auto-regressive policy head
- ❑ Ptr Net
- ❑ Centralized value baseline - COMA policy gradient

[IASI AI] Alphastar Model – Observation encoders & LSTM

- A **transformer** is used to do self-attention for encoding **sets of units** (Vaswani et al, NeurIPS 2017).
- This self-attention is used to add inductive biases to the architecture for learning relations between objects (Zambaldi et al, to be presented at ICLR 2019).
- The self-attention layers are combined with a deep **LSTM** to endow agent with memory
- A deep **ResNet** for encoding **points on the map**.
- A **scatter connection** from the transformer to the spatial encoder to combine spatial and non-spatial features.

Note: LSTM = deep LSTM, with 3 layers and 384 units each. This memory is updated every time AlphaStar acts in the game, and an average game takes about 1000 actions.

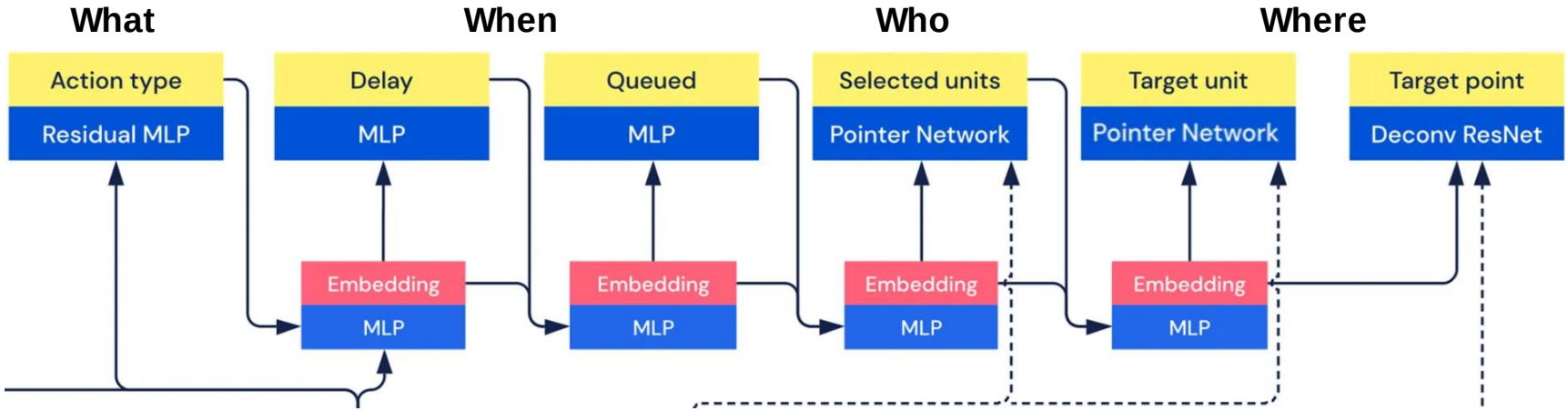
Note:
Units are like words and a set of units like a sentence so transformer is good for this.



Regarding scatter connections in supplementary material:
Two additional map layers are added to those described in the interface. The first is a camera layer with two possible values: whether a location is inside or outside the virtual camera.
The second is the scattered entities. `entity_embeddings` are embedded through a size 32 1D convolution followed by a ReLU, then scattered into a map layer so that the size 32 vector at a specific location corresponds to the units placed there.

An Overdue Post on AlphaStar

[IASI AI] Alphastar Model – Autoregressive action head



Autoregressive action head with 6 sub-heads:

- Four scalar heads: action type, action delay, action repeat, modifier key
- A recurrent pointer network to select a set of units by pointing to each unit in sequence [vinyals, Fortunato & Jaitiy, 2015]
- A simple pointer network to select single units
- A ResNet decoder to select points on the map

The policy is auto-regressive, meaning it predicts each action dimension conditionally on each previous action dimension. (Paper).
This also uses a **pointer network** (Vinyals et al, NeurIPS 2015), which more easily supports variable length outputs for variable length inputs. (Pointer Network: Use attention as pointer to input)
The agent is not deterministic. We sample an action at each time step from a policy head which assigns a non-zero probability to all possible actions.

How to sample from softmax with temperature , Deciphering AlphaStar on StarCraft II

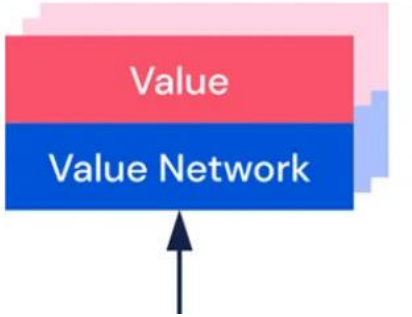
Note: CS285 Fa19 9/4/19 :

For imitation learning how to avoid averaging multimodal behaviour? Fow continuous high dimensional action space - use autoregressive discretization

[IASI AI] Alphastar Model – Observation encoders & LSTM

•The model then uses a **centralized value baseline**, linking a counterfactual policy gradient algorithm for multi-agent learning (Foerster et al, AAAI 2018).

• Used mainly for variance reduction during training, not needed on inference time

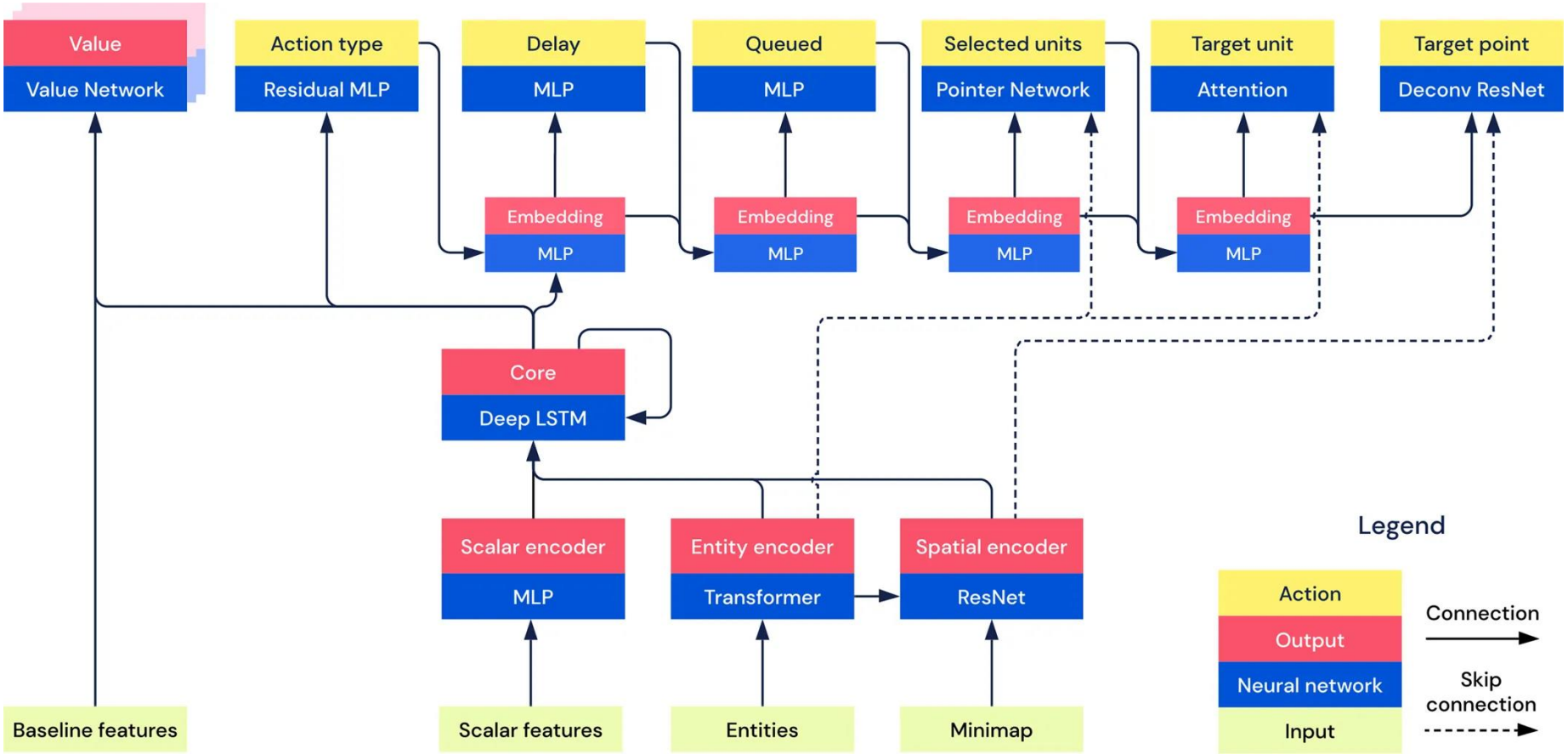


To reduce variance, during training only, the value function is estimated using information from both the player's and the opponent's perspectives.

Why reduce variance? We substract from the return of an experience the average return estimated by Value network. = Z-Scoring

An Overdue Post on AlphaStar

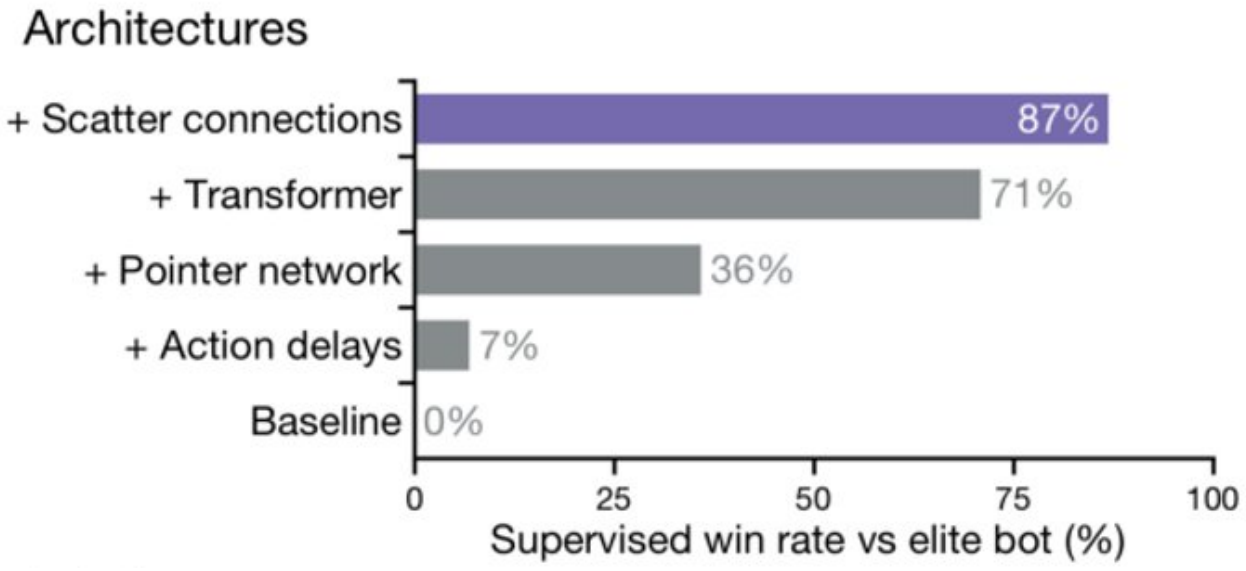
[IASI AI] Alphastar – Model arhitecture – training 140M , inference 70M parameters



Deciphering AlphaStar on StarCraft II

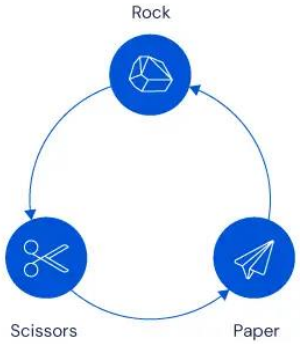
Extended Data Fig. 3: Overview of the architecture of AlphaStar.

[IASI AI] Alphastar – Model arhitecture – Key components ablation



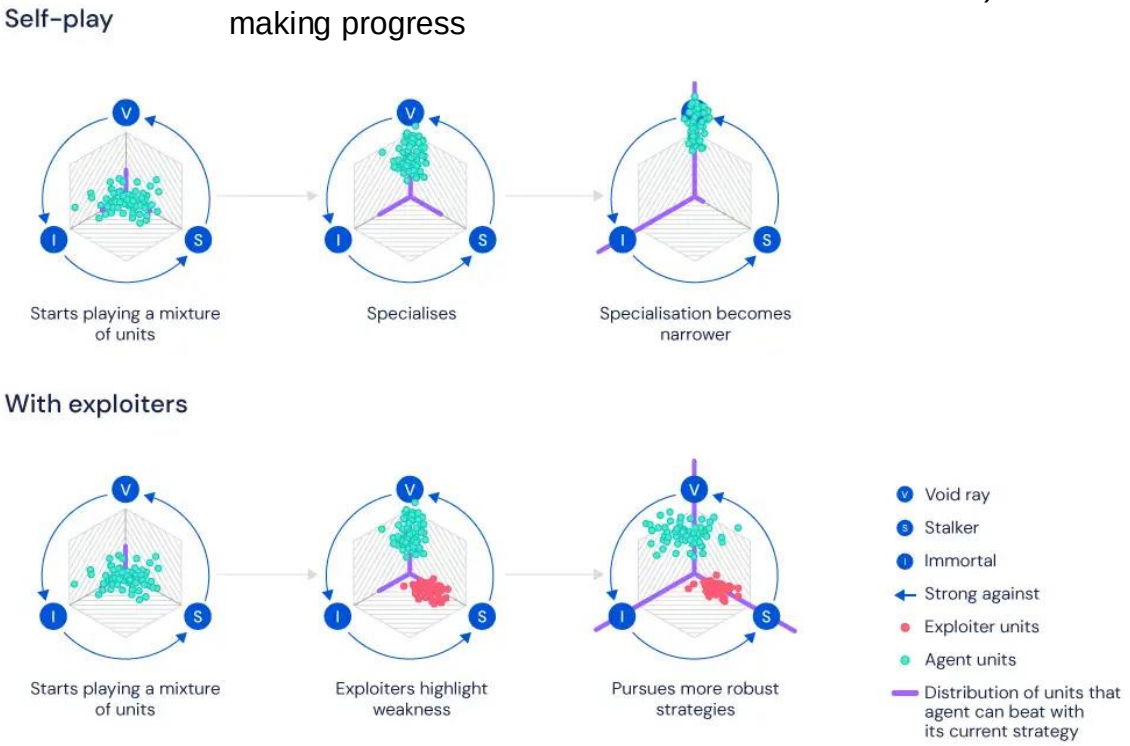
[IASI AI] Pure Self-play – Overfitting, Specialization

Self-play algorithms learn rapidly but may chase cycles (for example, where A defeats B, and B defeats C, but A loses to C) indefinitely without making progress



Rock, paper, scissors

StarCraft II players can create a variety of 'units', which have balanced strengths and weaknesses, similar to the game rock, paper, scissors



Deepmind blog

[IASI AI] League training explained

The League

- Multi-agent reinforcement learning algorithm
- Agents learn for a period and then are frozen
- New agents play current and past agents
- Addresses self-play cycles
- Diversifies strategies

Details:

- Form a continuous league wherein agents compete with each other
- New agents were dynamically supplemented to the league, by branching from existing competitors; each agent then learns from games against other competitors.
- Each agent would play against the strongest strategies and does not forget how to defeat the earlier version of agents.
- Put initialized agents into the league, and divide them into main exploiters, league exploiters and league exploiters.
  - main agents - against past players and themselves
  - main exploiters - against main agents
  - league exploiters - against all past players
- When adding a player to the league, reset main exploiters and league exploiters to supervised agents.
- Matchmaking strategies: prioritised Fictious Self-Play (pFSP)

[IASI AI] Fictitious Self Play & Prioritized Fictitious Self Play

Fictitious Self Play:
(= fight forgetting to defeat old strategies)

- Play against all previous players in the league
  - Uniform sampling of opponents

- Pro: Address issue with cycles/forgetting
- Con: Wastes games against easily defeated opponents

Prioritized Fictitious Self Play:
(= choose challenging opponents)

- Sample opponents strategically
- For agent A, sample opponent B from candidate set C with probability:

$$\frac{f(\mathbb{P}[A \text{ beats } B])}{\sum_{C \in \mathcal{C}} f(\mathbb{P}[A \text{ beats } C])}$$

The League - Main Agents

The champion.

- Goal: “Be the best StarCraft 2 agent in the League”
- 35% SP
- 65% PFSP against all past players
- Added after 2*10^9 steps
  (they are frozen and added to the league then new agent spawned)
- Never reset
  (to the supervised agent but spawned from most recent main agent)

The League - Main Exploiters

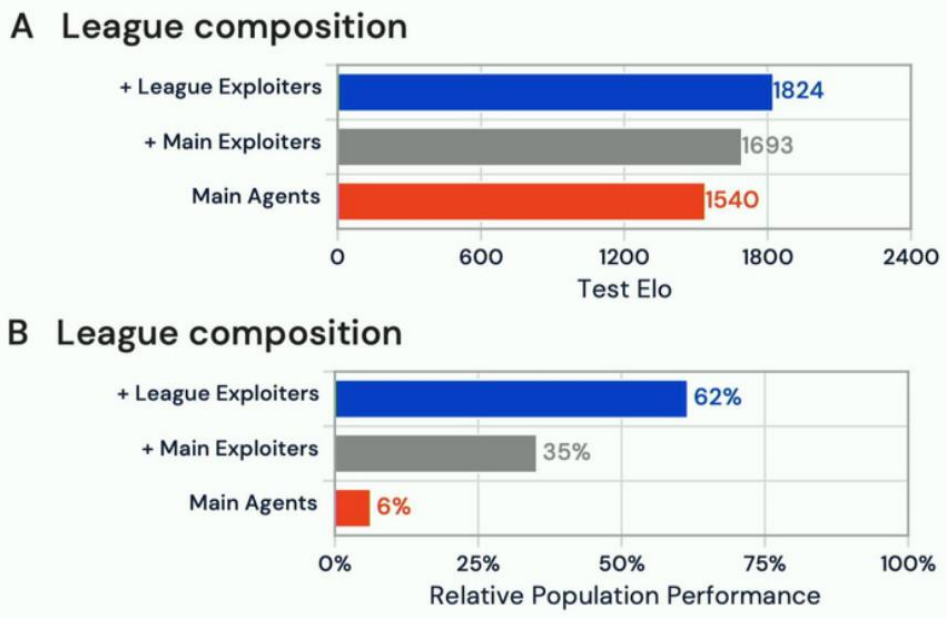
The sparring partner.

- Goal: “Highlight weaknesses in main agent strategies”
- 100% against main agents
- Added after successful or timeout
- Reset to supervised parameters

The coach.

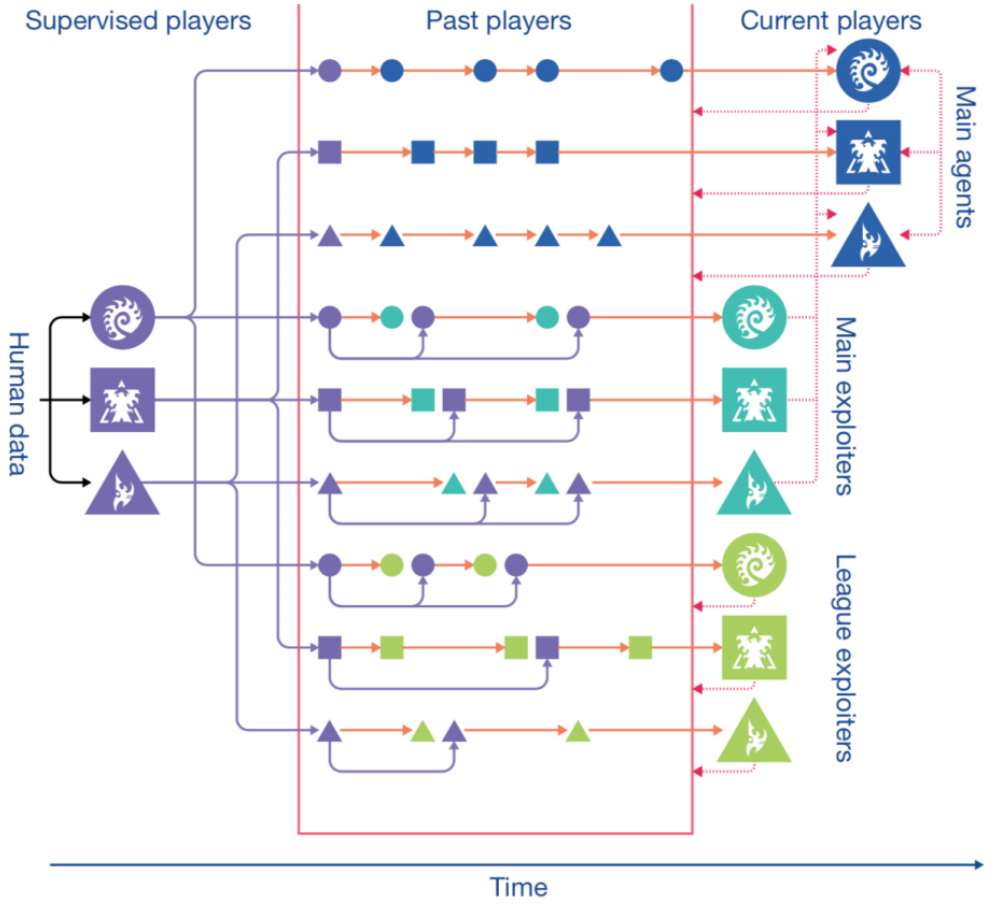
- Goal: “Highlight weaknesses in League strategies”
- 100% PFSP against all past players
- Added after successful or timeout
- Reset to supervised parameters with P=0.25

League training - Ablations



Elo score against held-out agents (exploiters)

[IASI AI] League training explained



- Build strong, robust agents

They keep training and playing against the league, sometimes they drop a checkpoint

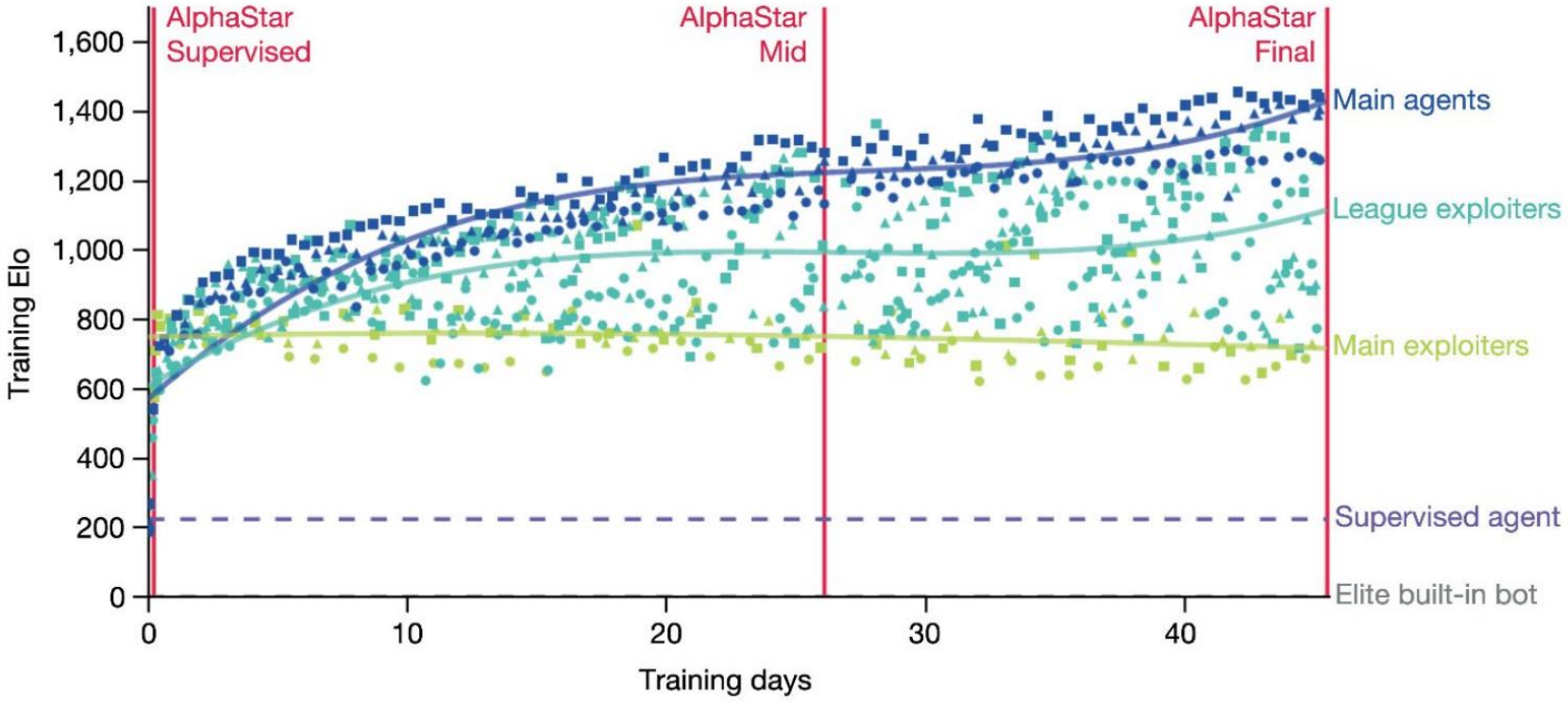
- Expose main agents weaknesses:

Their purpose is to beat main agents (main agents params are fixed until later)

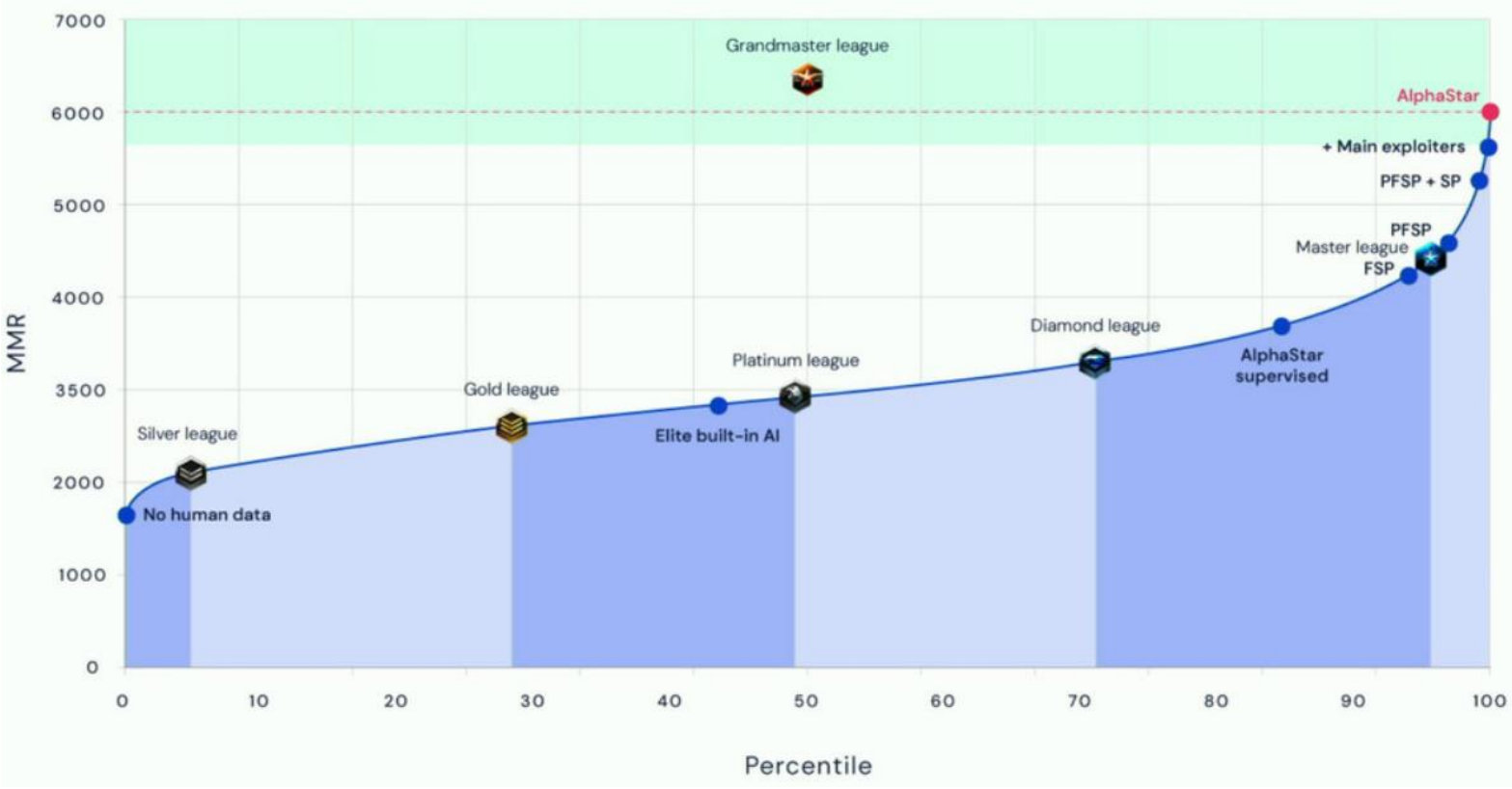
Matchmaking target
Reinforcement learning
Supervised learning
Reinitialization

Zerg
Terran
Protoss

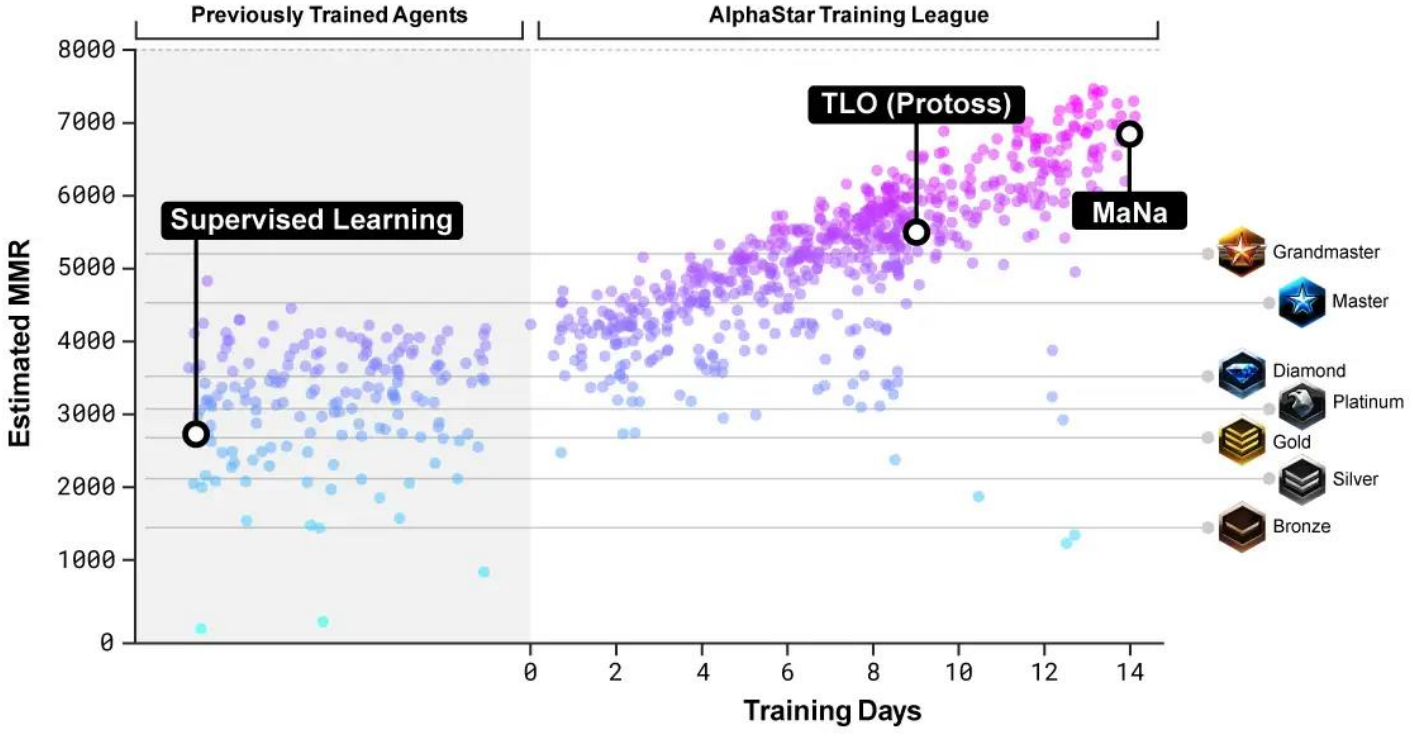
[IASI AI] Training Results



[IASI AI] Results by playing anonymously on Battlenet against human players



[IASI AI] Alphastar Training – Population based + Multi agent - fight overfitting/ forgetting

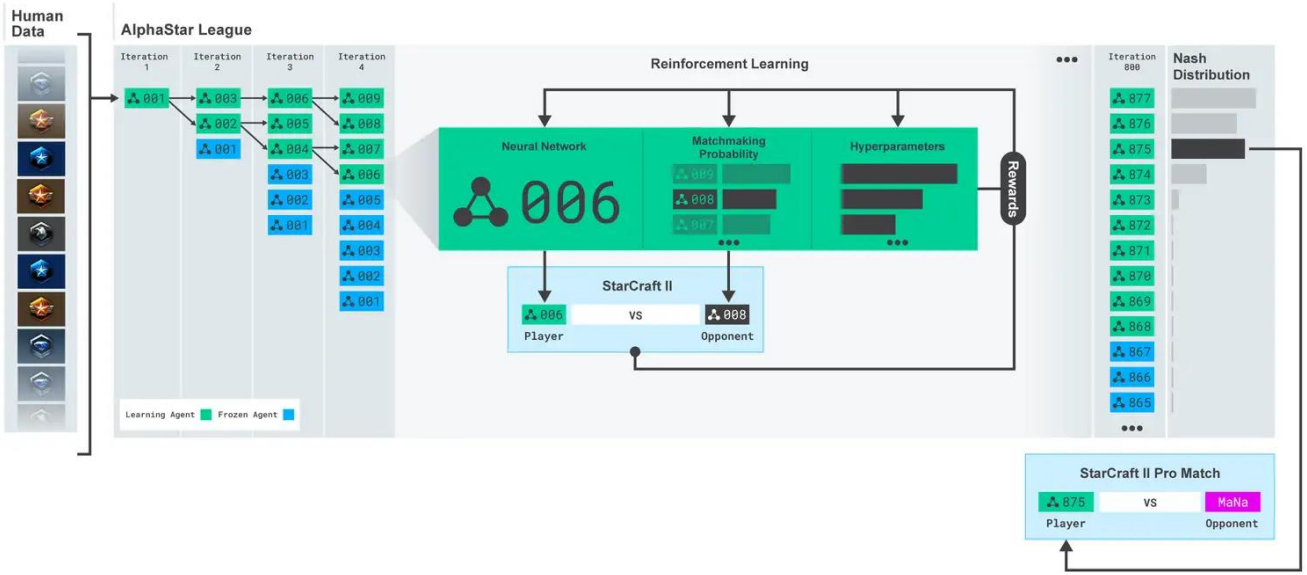


Estimate of the match making rating (mmr) - an approximate measure of a player's skill - for competitors in the alphastar league, throughout training, in comparison to blizzard's online leagues.

AlphaStar: Mastering the Real-Time Strategy Game StarCraft II

[IASI AI] Alphastar League Training

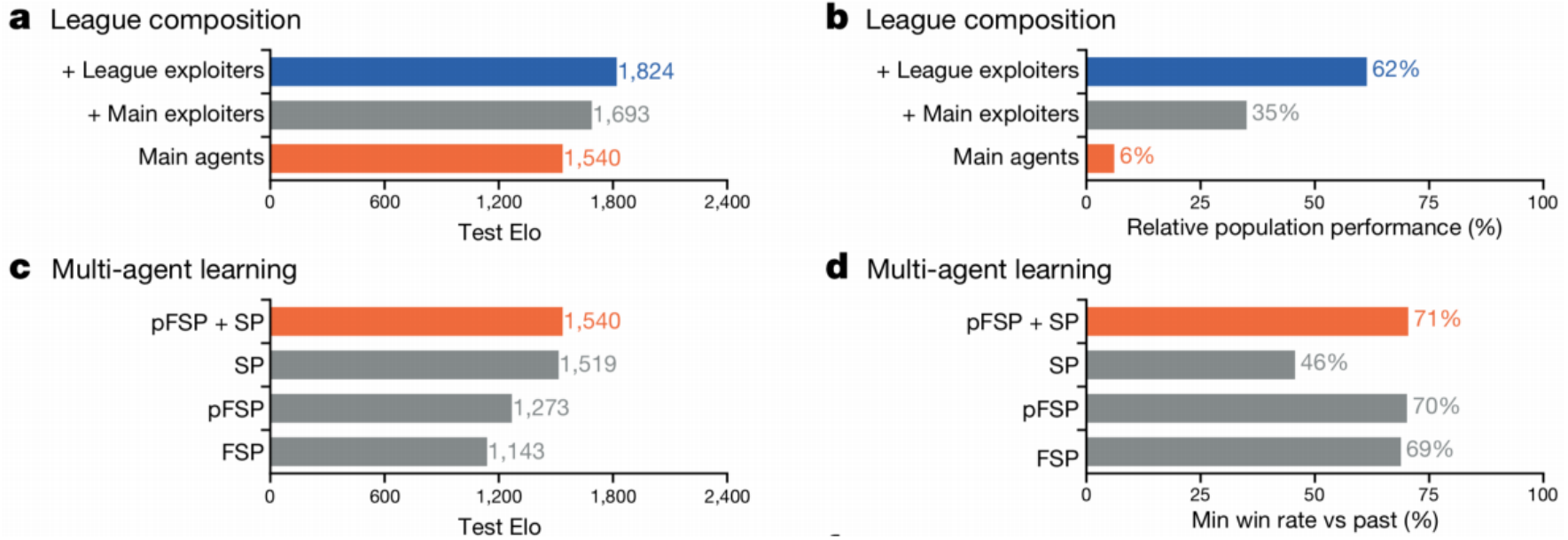
The league – for fighting overfitting to one strategy / forgetting old strategies



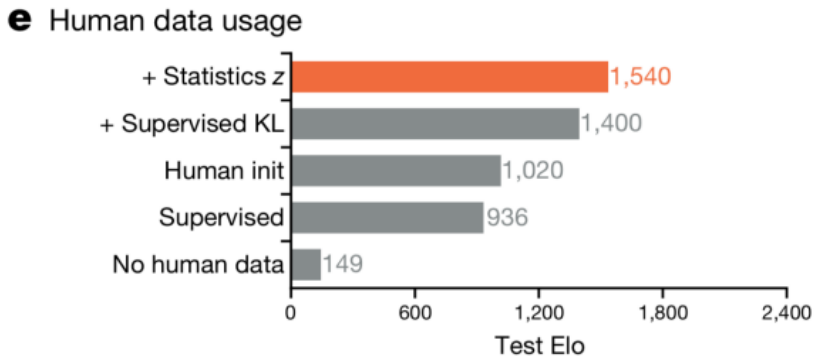
Agents are initially trained from human game replays, and then trained against other competitors in the league. At each iteration, new competitors are branched, original competitors are frozen, and the matchmaking probabilities and hyperparameters determining the learning objective for each agent may be adapted, increasing the difficulty while preserving diversity. The parameters of the agent are updated by reinforcement learning from the game outcomes against competitors. The final agent is sampled (without replacement) from the nash distribution of the league.

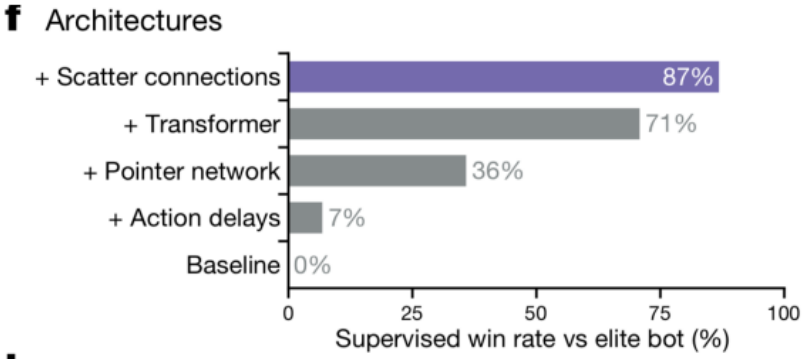
AlphaStar: Mastering the Real-Time Strategy Game StarCraft II, Capture the Flag: the emergence of complex cooperative agents, A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning

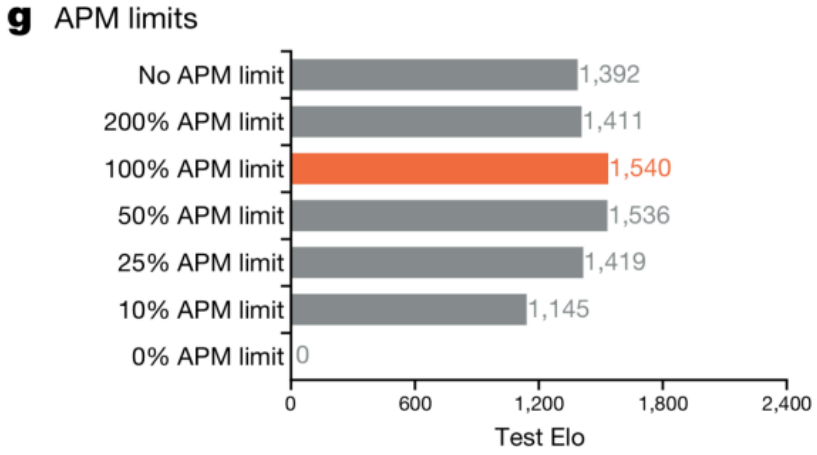
[IASI AI] Ablations for key components of AlphaStar

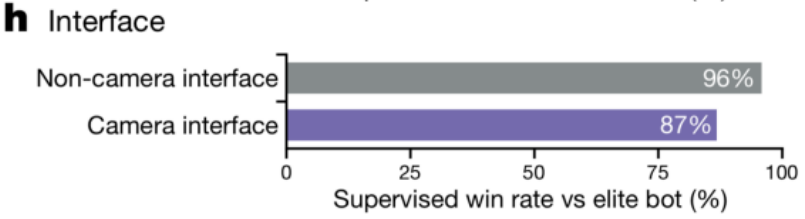


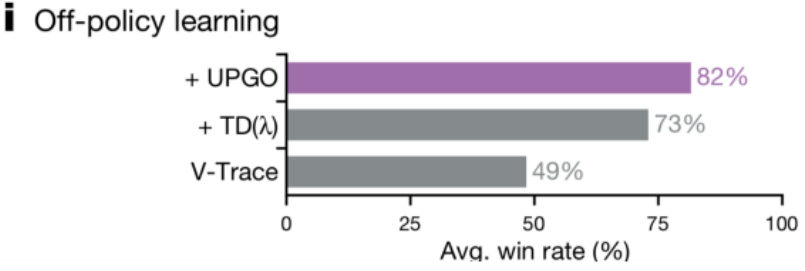
[IASI AI] Ablations for key components of AlphaStar











[IASI AI] Lessons learned

1. **AlphaStar has achieved Grandmaster level at the full game of StarCraft 2**

   a. Input: state; output: action → **Model Free**!

   b. without any scripting / rule based sub-system

   c. With all races, a camera view, and pro approved interface

2. **Imitation learning** yields a strong agent

3. The AlphaStar League produces **robust** agents

   a. Cannon rush, probe rush, early attacks, many unit compositions, ...

[IASI AI] Challenges in Deep RL/ML

- Deep Learning General Purpose Architectures
  - Increased capabilities to cope with images, sequences, sets, subsets, pointers
- Exploration
  - More work towards better / advanced imitation
  - Imitation is a patch. More progress and ideas needed!
- Robustness
  - League is v0 to fix robustness. More methods should be researched!

[IASI AI] Challenges in Deep RL/ML

- Real world problems
  - OK to use simulation (but hard to simulate real world!)
- Transfer / General AI
  - AlphaGo should learn Chess quickly
  - AlphaStar should learn ATARI easily
    - And should learn from every game it plays!
  - ImageNet classifier should trivially transfer to MNIST
- Understanding & Theory: non-convex, non-stationary, multi-agent

[IASI AI] Alphastar AI – DeepMind AlphaStar, StarCraft, and Language Podcast

Starcraft 2:
– real-time, partial obs = imperfect information, long term planning, large action space, many units, variety of strategies possible, continuous actions
- predict enemy intentions – need to decide scouting – all implicit learned
Biggest pb - exploration: start learning + long term planning via random exploration is impossible
- the machine does not understand the world as we are
- partial observability – you need memory – you need LSTM to remember and use information from past observations
- population of agents trained against older versions leads to diversity of personalities which leads to robust strategies
Architecture – policy net (transformer + LSTM)
inputs = zoomed in minimap, zoomed out screen, summary of game state = list of units it can see and control
Given all prev partial obs and actions predict next action = NLP pretraining for Lang Generation (supervised training)
Imitiation (like translation code)
- imitation learning + control bit input for the level of imitated human player

League agents – they developed cheesy strategies
Population based training (League) - better than self-play – diversity
What's next:
- meta-learn – adapt fast to play another race (not only Protoss), bluffing?, theory of mind?

DeepMind's AlphaStar (Vinyals et al, 2019)
(Grandmaster level in StarCraft II using multi-agent reinforcement learning)

- based on the FTW setup used to tackle Quake III: Combine a **distributed IMPALA actor-learner setting with a powerful prior that induces structured exploration:**

While FTW uses a prior based on a time-scale hierarchy of two LSTMs, AlphaStar makes use of human demonstrations.

- **prioritized fictitious self-play** (aka *The League*),
- an autoregressive decomposition of the policy with pointer networks
- **upgoing policy update** (UPGO - an evolution of the V-trace Off-Policy Importance Sampling correction for structured action spaces)
- **scatter connections** (a special form of embedding that maintains spatial coherence of the entities in map layer)

[IASI AI] Links - Alphastar AI

Talks:

Grandmaster Level in StarCraft II using Multi-Agent Reinforcement Learning (NeurIPS2019 talk)

Deepmind's AlphaStar paper explained (Slides)

Offcial Blog:

AlphaStar: Grandmaster level in StarCraft II using multi-agent reinforcement learning

AlphaStar: Mastering the Real-Time Strategy Game StarCraft II

Paper:

Grandmaster level in StarCraft II using multi-agent reinforcement learning (Link2)

Grandmaster level in StarCraft II using multi-agent reinforcement learning (unformatted)

From Alphastar creators:

Johannes Daub: A new Star is born - Looking into AlphaStar

AlphaStar: Mastering the Real-Time Strategy Game Starcraft II (Slides)

Oriol Vinyals: DeepMind AlphaStar, StarCraft, Language, and Sequences

Older:

Ongoing survey on StarCraft research

[IASI AI] Links - Alphastar AI

Blog – Alphastar Deciphered:
Deciphering AlphaStar on StarCraft II (Very technical + Slides also)
An Overdue Post on AlphaStar
DeepMind's AI, AlphaStar Showcases Significant Progress Towards AGI
What DeepMind's AlphaStar beating StarCraft players means for AI research

Reddit:
Q&A with Oriol Vinyals and David Silver from DeepMind's AlphaStar team
Comments for the Paper
What can DeepMind's AlphaStar tell us about imitation learning for self-driving cars?

DeepMind's "AlphaStar" StarCraft 2 demonstration livestream

Code:
Alphastar Pseudocode , Population Based Training (in PyTorch with sqlite3)

Demo:
DeepMind challenges Starcraft 2 Pro - AlphaStar (AI) vs MaNa
How AlphaStar Became a StarCraft Grandmaster

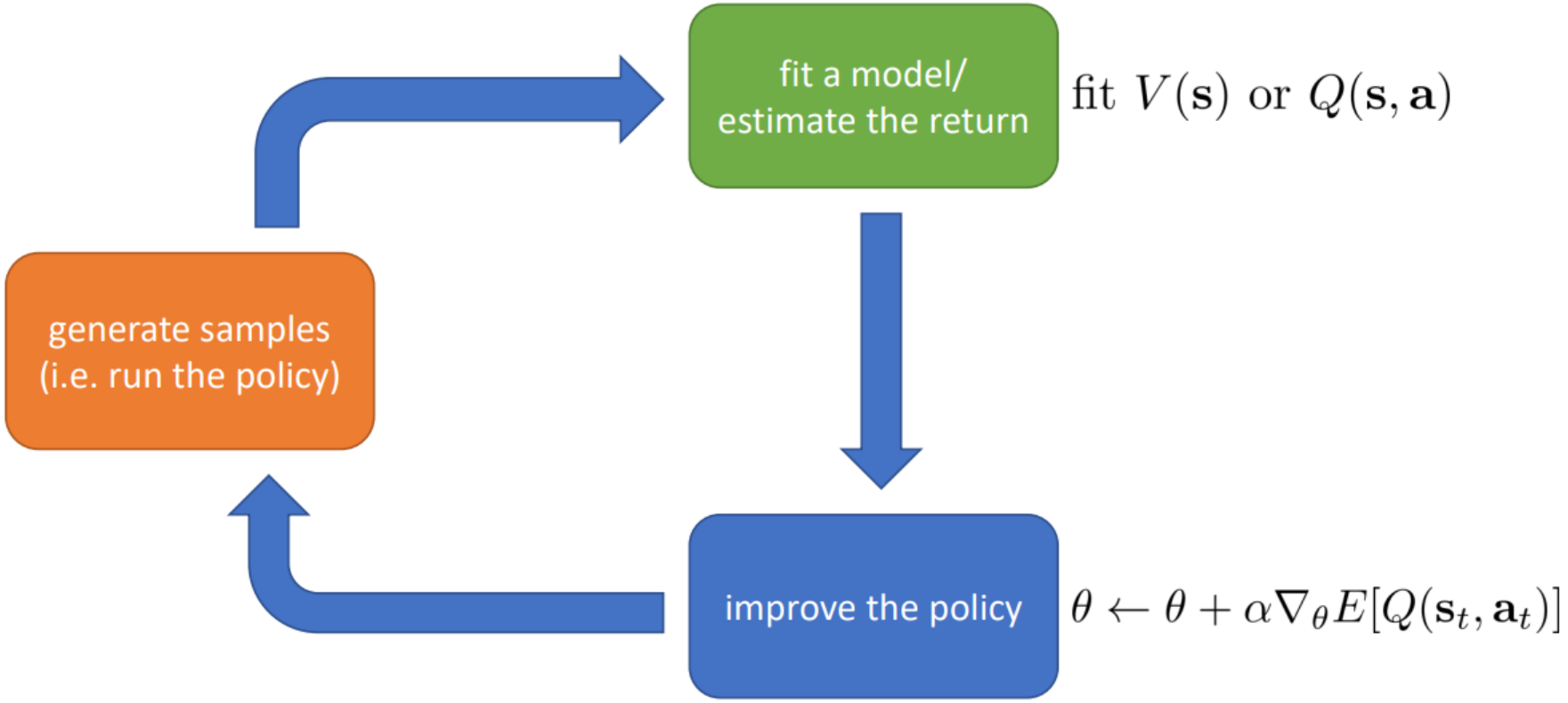
[FORTECH.AI]

[IASI AI]

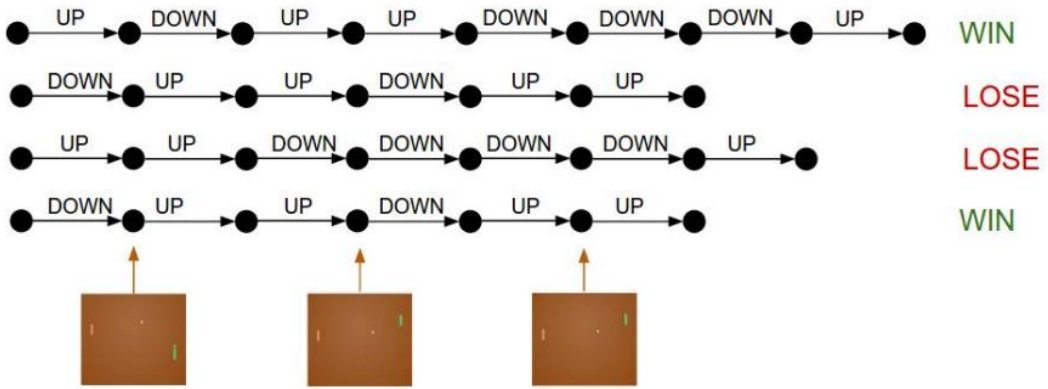
Related papers used to implement Alphastar DRL Agent

Alex Movila

Actor-critic: value functions + policy gradients



[IASI AI] Policy gradient – sampling episode



Trajectory: $\tau = \{(s_1, a_1), (s_2, a_2), \ldots, (s_T, a_T)\}$

Reward: $R(\tau) = \sum_{t=0}^{|\tau|} R_t \quad ; R_t \equiv r(s_t, a_t)$

$$\max_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

Gt in our case ("return")

[IASI AI] Policy gradient - update

$$\theta \leftarrow \theta + \lambda \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

$$\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

$$= \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{|\tau|} R_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

← Gradient of weighted log-likelihood

Policy gradient - math to implement update

Formulate the gradient in terms of policy instead of rewards

$$\nabla_\theta \mathbb{E}[R(\tau)] = \nabla_\theta \Big(\sum_{t=0}^{|\tau|} \pi_\theta(a_t|s_t) R_t\Big)$$

$$= \sum_{t=0}^{|\tau|} (\nabla_\theta \pi_\theta(a_t|s_t) R_t)$$

Log-trick

$$= \sum_{t=0}^{|\tau|} (\pi_\theta(a_t|s_t) \nabla_\theta \log \pi_\theta(a_t|s_t) R_t)$$

$$= \mathbb{E}_{\tau \sim \pi_\theta}\Big[\sum_{t=0}^{|\tau|} R_t \nabla_\theta \log \pi_\theta(a_t|s_t)\Big]$$

Log-trick:

$$\pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau)$$

$$= \pi_\theta(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)}$$

$$= \nabla_\theta \pi_\theta(\tau)$$

Reward:

$$R(\tau) = \sum_{t=0}^{|\tau|} R_t$$

Note: it is better to output log probability from NN because it is unconstrained in (0,1)

Temporal Difference (TD) Error and TD(0)

Observe samples $< s_t, a_t, r_t, s_{t+1} >$. If value estimates are accurate, the following must hold:

$$V(s_t) = r_t + \gamma V(s_{t+1})$$

If not, there is an error (TD error):

$$\delta = r_t + \gamma V(s_{t+1}) - V(s_t)$$

To learn better estimates – minimize δ (TD(0)):

$$V(s) \leftarrow V(s) + \alpha\big(r_t + \gamma V(s_{t+1}) - V(s_t)\big)$$

[Samuel 1959; Sutton 1984, 1988]

[IASI AI] Alphastar policy update - variation of advantage actor-critic

RL policy update: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)},$

Value network ("critic") estimates V. Updated at same time

$$\nabla_\theta J(\theta) \sim \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\left(r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)\right)$$

Policy network ("actor") is updated by this gradient

This part is modified in the paper to V-trace

This part is modified in the paper to TD-lambda

[IASI AI] Temporal difference – Value estimation

TD learning can learn from **incomplete** episodes of experience and hence we don't need to track the episode up to termination.

TD learning methods update targets with regard to existing estimates rather than exclusively relying on actual rewards and complete returns. (**bootstrapping**)

The key idea in TD learning is to update the value function $V(S_t)$ towards an estimated return $R_{t+1} + \gamma V(S_{t+1})$ (known as "**TD target**"). To what extent we want to update the value function is controlled by the learning rate hyperparameter α:

$$V(S_t) \leftarrow (1-\alpha)V(S_t) + \alpha G_t$$
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$
$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

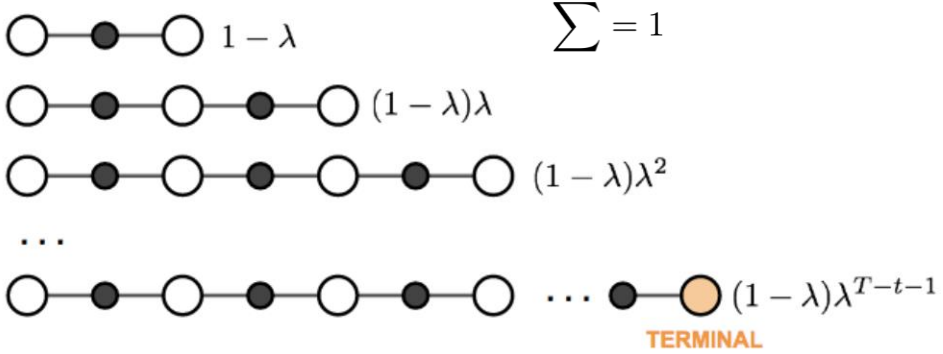
Reinforcement Learning — TD(λ) Introduction
Combining TD and MC Learning

[IASI AI] Temporal difference – TD(lambda)

n	G_t	Notes
$n = 1$	$G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$	TD learning
$n = 2$	$G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$	
…		
$n = n$	$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$	
…		

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(n)} - V(S_t))$$

At time step t, $\mathrm{TD}(\lambda)$



$$\begin{aligned} \text{let } S &= 1 + \lambda + \lambda^2 + \ldots \\ S &= 1 + \lambda(1 + \lambda + \lambda^2 + \ldots) \\ S &= 1 + \lambda S \\ S &= 1/(1-\lambda) \end{aligned}$$

$$\lambda\text{-return } G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}.$$

Reinforcement Learning — TD(λ) Introduction
Combining TD and MC Learning

[IASI AI] TD(lambda) - Loss function using Off-line λ-Return

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^{n} \hat{v}(S_{t+n}, \mathbf{w}_{t+n-1}), \quad 0 \leq t \leq T-n,$$

$$G_t^{\lambda} \quad = \quad (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} \quad + \quad \lambda^{T-t-1} G_t,$$

$$\begin{aligned} \mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2} \alpha \nabla \Big[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t) \Big]^2 \\ &= \mathbf{w}_t + \alpha \Big[v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t) \Big] \nabla \hat{v}(S_t, \mathbf{w}_t), \end{aligned}$$

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \Big[G_t^{\lambda} - \hat{v}(S_t, \mathbf{w}_t) \Big] \nabla \hat{v}(S_t, \mathbf{w}_t), \quad t = 0, \ldots, T-1.$$

Reinforcement Learning — TD(λ) Introduction

[IASI AI] TD(lambda) - even better than N-step return

TD(λ)

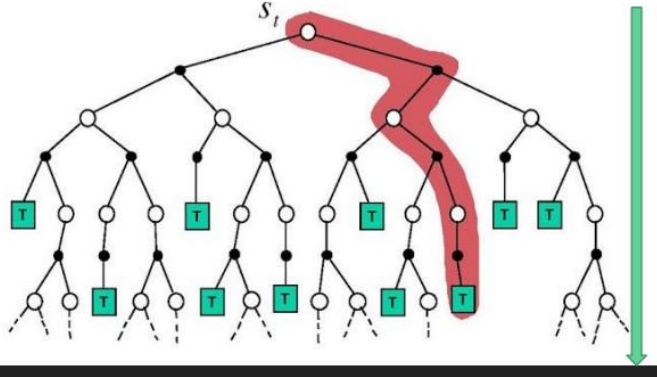
$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^{n} V_{t+n-1}(S_{t+n}),$$

full episode ("Monte Carlo")

one step

Monte-Carlo Backup

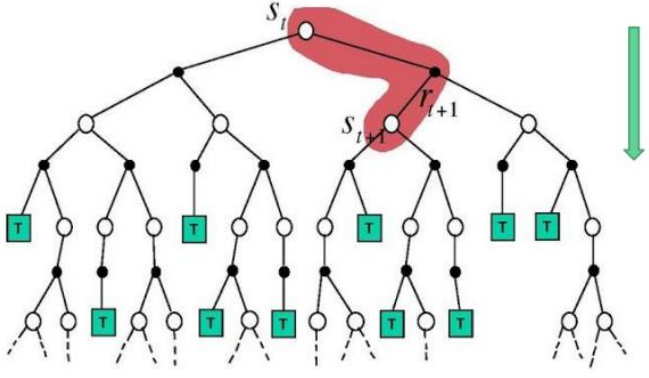
$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$$



n="∞"
(λ=1)

Temporal-Difference Backup

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$

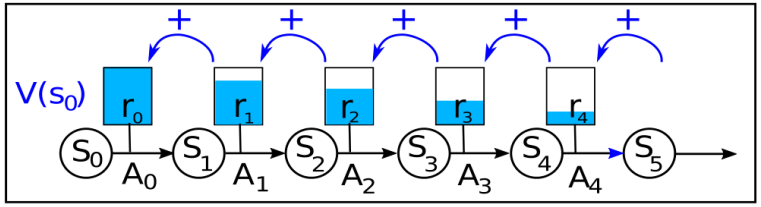


n=1
(λ=0)

VARIANCE ⟷ BIAS

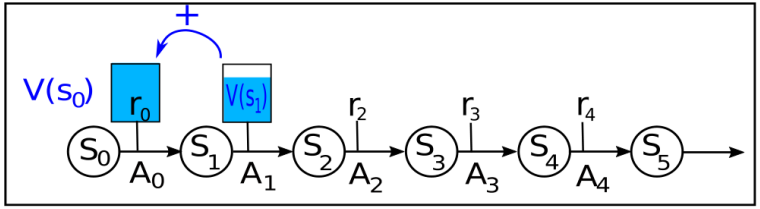
TD-λ is "in between"

[IASI AI] Monte Carlo, One-step TD and N-step TD - Issues



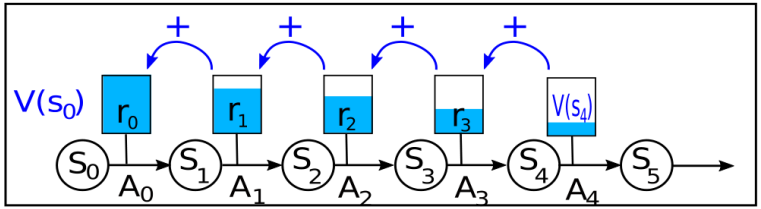
Monte Carlo

- MC suffers from variance due to exploration of a long trajectory (+ stochastic trajectories)
- MC is on-policy → less sample efficient



One-step TD

- One step TD suffers from (overestimation) bias



N-step TD

- N-step TD: tuning N to control the bias variance compromise

TD(lambda) Methods - Generalized Advantage Estimation

- Recall, finite-horizon advantage estimators

$$\hat{A}_t^{(k)} = r_t + \gamma r_{t+1} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k}) - V(s_t)$$

- Define the TD error $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$
- By a telescoping sum,

$$\hat{A}_t^{(k)} = \delta_t + \gamma \delta_{t+1} + \cdots + \gamma^{k-1} \delta_{t+k-1}$$

- Take exponentially weighted average of finite-horizon estimators:

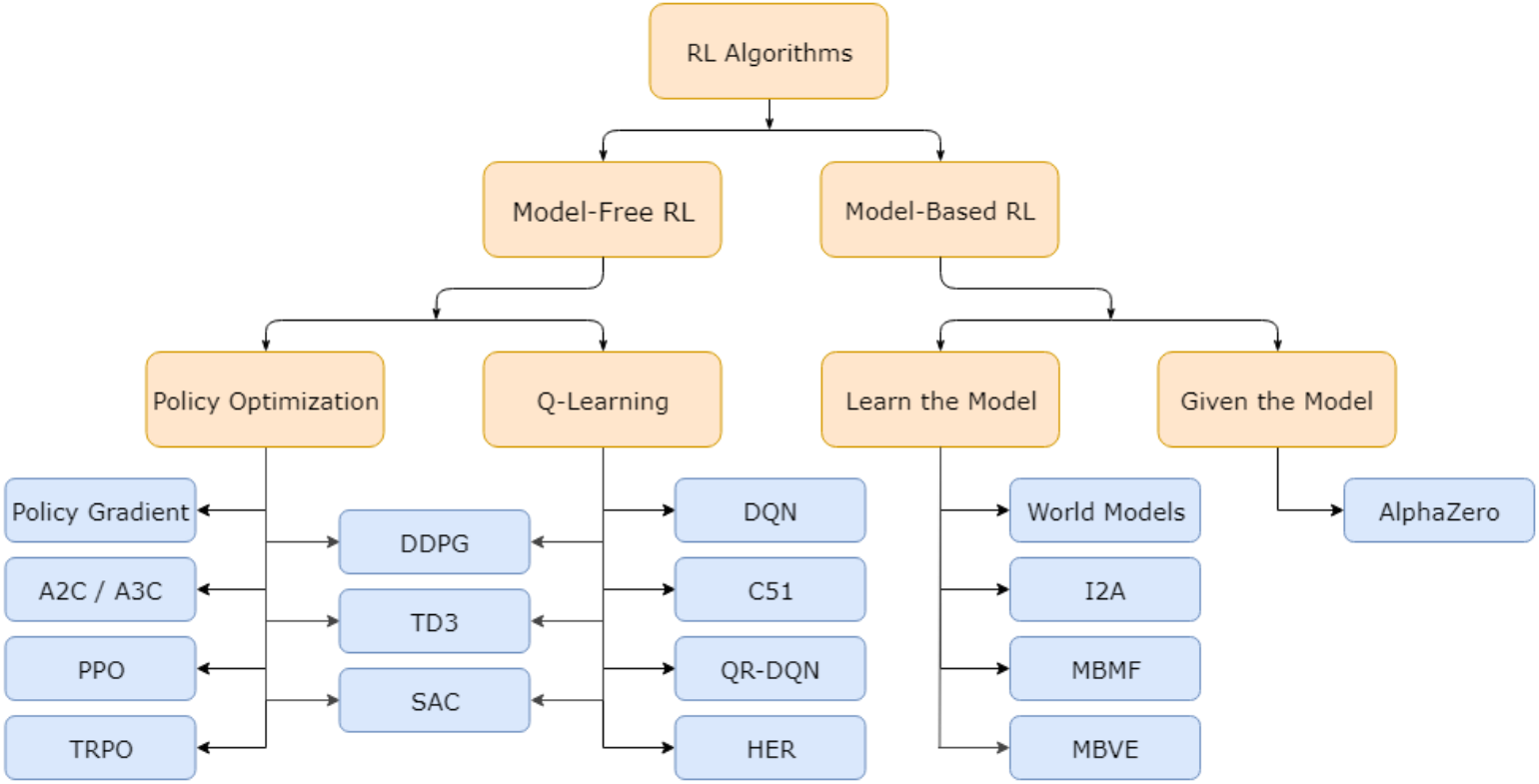
$$\hat{A}^{\lambda} = \hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \ldots$$

- We obtain

$$\hat{A}_t^{\lambda} = \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2 \delta_{t+2} + \ldots$$

- This scheme named *generalized advantage estimation* (GAE) in [1], though versions have appeared earlier, e.g., [2]. Related to TD(λ)

[IASI AI] Deep RL algorithms



Alphastar uses an improved, scalable version of A3C named IMPALA

Asynchronous Advantage Actor-Critic (A3C)

A3C [Mnih et al., 2016] is an asynchronous actor-critic algorithm

- Learn state-value function $V^{\pi}(x) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | x, \pi\right]$ by minimizing a n-step Temporal Difference:

$$\left(V_w(x_t) - \left(\underbrace{r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n V_w(x_{t+n})}_{n-\text{steps target value}}\right)\right)^2$$

- Policy $\pi_\theta(a_t|x_t)$ improved by following gradient ascent:

$$\nabla \log \pi_\theta(a_t|x_t)\left(r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n V_w(x_{t+n}) - V_w(x_t)\right)$$

Properties: On-policy algorithm, uses multi-steps learning.

Asynchronous Advantage Actor-Critic (A3C)

- A2C / A3C uses this fixed-horizon advantage estimator
- Pseudocode

for iteration$=1, 2, \ldots$ **do**

Agent acts for T timesteps (e.g., $T = 20$),

For each timestep t, compute

$$\hat{R}_t = r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_t)$$

$$\hat{A}_t = \hat{R}_t - V(s_t)$$

$\hat{R}_t$ is target value function, in regression problem

$\hat{A}_t$ is estimated advantage function

Compute loss gradient $g = \nabla_\theta \sum_{t=1}^{T} \left[-\log \pi_\theta(a_t \mid s_t) \hat{A}_t + c(V(s) - \hat{R}_t)^2 \right]$

g is plugged into a stochastic gradient descent variant, e.g., Adam.

end for

DQN versus A3C

DQN:

- Pros: Off-policy learning use memory replay (sample efficiency), allow any exploration strategy
- Cons: One-step learning: Slow to propagate information, accumulates errors, no RNNs

A3C:

- Pros: Multi-steps learning fast propagation of information, possible use of RNNs
- Cons: On-policy learning: does not allow memory replay, neither exploration

The Need: off-policy, multi-steps learning

Two desired properties of a RL algorithm

- Off-policy learning:

  - use memory replay
  - do exploration
  - lag between acting and learning

- Use multi-steps learning:

  - propagate rewards rapidly
  - avoid accumulation of approximation/estimation errors
  - allow learning from sequences (RNN)

Ex: Q-learning (and DQN) is off-policy but does not use multi-steps returns Policy gradient (and A3C) use returns but are on-policy.

Both properties are important in deepRL. Can we have both simultaneously?

[IASI AI]

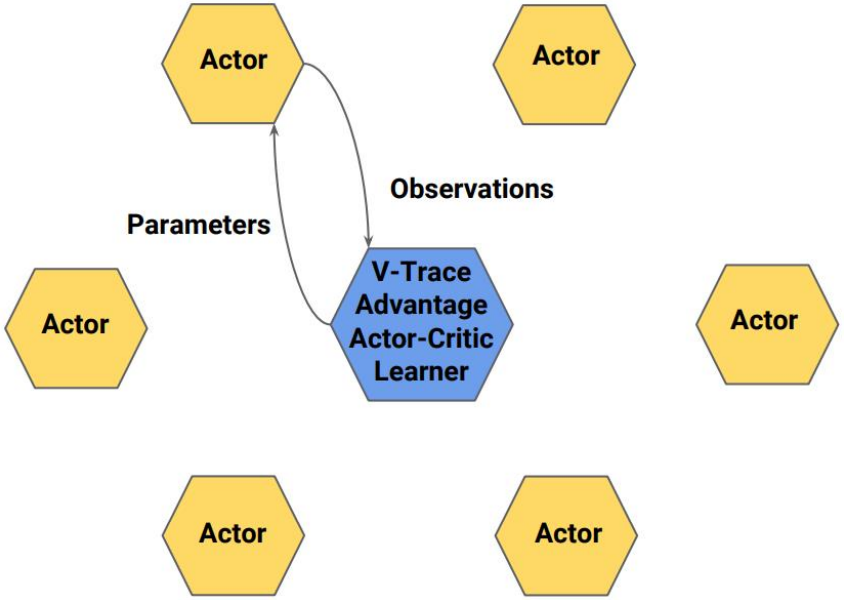
IMPALA [Espeholt et al., 2018]

IMPortance Weighted **A**ctor-**L**earner **A**rchitecture

- Heavily distributed architecture
- Many actors (CPU),
- One (or more) learner (GPU)
- Actors generate trajectories and place them into a queue.
- Learner dequeues and performs parameter updates.

Problem: Stale experience → requires off-policy learning: V-trace

IMPALA proved to achieve better performance compared to A3C variants in terms of data efficiency, stability and final performance.
(10x more data efficient, 2x overall final performance)



IMPALA - Pro: no suffering from variance in frame rendering-time or time consuming task restarts

Cons: decoupling the acting and learning causes the policy in the actor to lag behind the learner

Scalable agent architecture for distributed training (Code)

IMPALA - **Im**portance Weighted **A**ctor-**L**earner **A**rchitecture vs A3C

IMPALA proved to achieve better performance compared to A3C variants in terms of data efficiency, stability and final performance.

System Config:

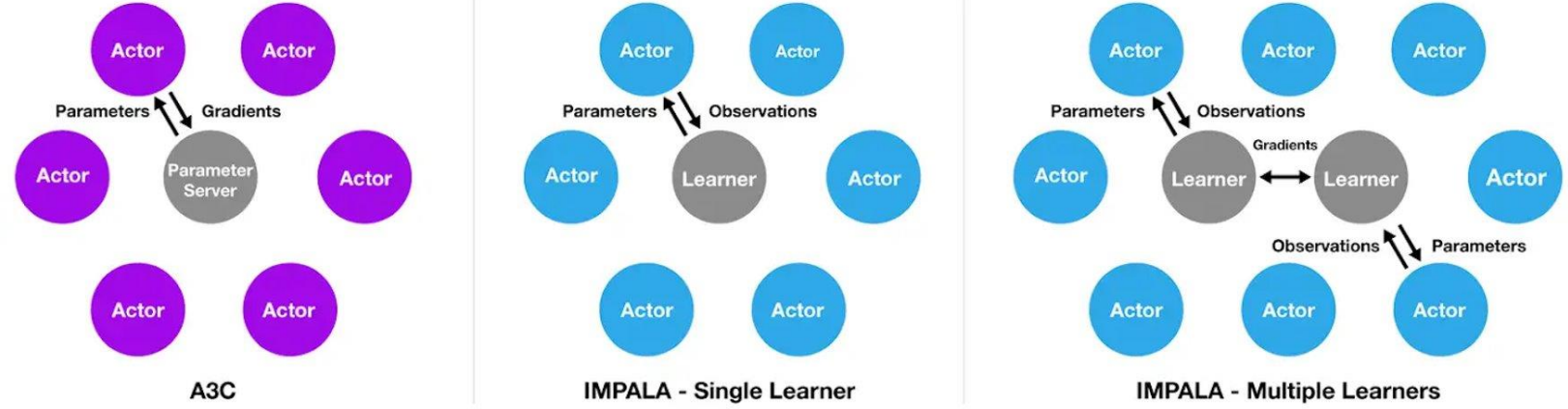
- "Actors" run asynchronously on distributed CPU resources (cheap)
- "Learner" runs on GPU; batched experiences received from actors
- Actors generate trajectories and place them into a queue.
- Learner dequeues and performs parameter updates.
- Actors periodically receive new parameters from learner

Algorithm:

- Policy gradient algorithm: descended from A3C
- Policy lag mitigated through off-policy learning with V-trace algorithm ("Importance Weighted")

Scale:

- Hundreds of actors (cheap on CPU), can use multi-GPU learner
- (learned all 57 games simultaneously; speed not reported)



A3C:

Periodically, actors pause their exploration to share the gradients they have computed with a central parameter server that applies updates

Actors are just used to collect experience which is passed to a central learner that computes gradients, resulting in a model that has completely independent actors and learners

[IASI AI]

IMPALA – V-trace actor-critic algorithm

V-trace actor-critic algorithm

Consider parameterized representations of the policy π_θ and value function V^π_ϕ. Trajectories have been generated by actors using the behaviour policy μ_θ. We denote the **V-trace targets** as v_t. At training time t, the value parameters ϕ are updated by gradient descent on the MSE loss to the target v_t, i. e. in the direction of:

use MSE loss to update Value network to V-trace target

$$\left(v_t - V^\pi_\phi(s_t)\right)\nabla_\phi V^\pi_\phi(s_t)$$ Gradient Descent of MSE

and the policy parameters θ are updated in the direction of the policy gradient:

$$\rho_t \nabla_\theta \log \pi_\theta(a_t|s_t)\left(r_t + \gamma v_{t+1} - V^\pi_\phi(s_t)\right)$$

An additional **entropy regularization bonus** may be added to promote exploration and prevent premature convergence:

add pseudo reward for high entropy of action distribution

$$-\nabla_\theta \sum_a \pi_\theta(a|s_t) \log \pi_\theta(a|s_t)$$

The overall update rule is obtained by summing these gradients scaled by appropriate coefficients (which are hyperparameters of the algorithm).

IMPALA: Scalable Distributed Deep-RL, Recent Achievements in Large-Scale Deep RL, Pantic-IMPALA .ppt , off-policy deep RL

[IASI AI]

IMPALA – V-trace actor-critic algorithm (n-step TD + imp sampling)

Definition (**V-trace target for $V^{\pi}_{\phi}(s_t)$**)

Consider a trajectory $\tau = (s_t, a_t, r_t)_{t=s,\dots,s+n}$ generated by the actor following some policy μ_θ. We define the n-step V-trace target for $V^{\pi}_{\phi}(s_s)$, our value approximation at state s_s as:

Sum up all temporal diffs across the trajectory => we want them pushed to zero + weight them differently according to importance

$$v_s \overset{\text{def}}{=} V^{\pi}_{\phi}(s_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s} \Big(\prod_{i=s}^{t-1} c_i \Big) \delta_t V^{\pi}_{\phi}$$

Multiply Ci imp. sampl. Weights up to crt step – like probabilities for each previous step

where $\delta_t V^{\pi}_{\phi} \overset{\text{def}}{=} \rho_t \big(r_t + \gamma V^{\pi}_{\phi}(s_{t+1}) - V^{\pi}_{\phi}(s_t) \big)$ is a **temporal difference** for V^{π}_{ϕ}, and $\rho_t \overset{\text{def}}{=} \min(\overline{\rho}, \frac{\pi_\theta(a_t|s_t)}{\mu_\theta(a_t|s_t)})$ and $c_i \overset{\text{def}}{=} \min(\overline{c}, \frac{\pi_\theta(a_i|s_i)}{\mu_\theta(a_i|s_i)})$ are truncated **importance sampling weights**. We assume $\prod_{i=s}^{t-1} c_i = 1$ for $s = t$ and $\overline{\rho} \geq \overline{c}$. The formula for the V-trace target can be rewritten recursively:

$$v_s = V^{\pi}_{\phi}(s_s) + \delta_s V^{\pi}_{\phi} + \gamma c_s \big(v_{s+1} - V^{\pi}_{\phi}(s_{s+1}) \big)$$

Key idea: downweight TD learned value from trajectories of misbehaving actors

V-trace can be inefficient in large, structured action spaces like the one we used for StarCraft, because distinct actions can result in similar (or even identical) behaviour.

Off-policy learning & importance sampling

$$\theta^\star = \arg\max_\theta J(\theta)$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

what if we don't have samples from $\pi_\theta(\tau)$?

(we have samples from some $\bar{\pi}(\tau)$ instead)

$$J(\theta) = E_{\tau\sim\bar{\pi}(\tau)}\left[\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)} r(\tau)\right]$$

$$\pi_\theta(\tau) = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$$

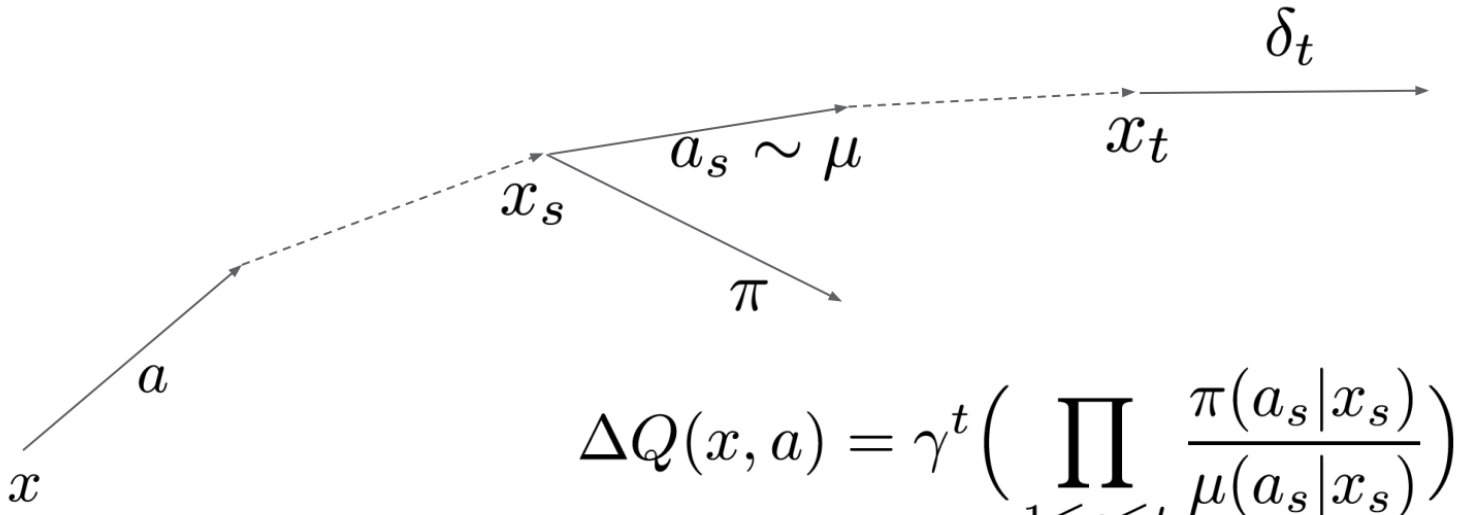
$$\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)} = \frac{\cancel{p(\mathbf{s}_1)}\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}}{\cancel{p(\mathbf{s}_1)}\prod_{t=1}^{T}\bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)\cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}} = \frac{\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^{T}\bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)}$$

importance sampling

$$\begin{aligned} E_{x\sim p(x)}[f(x)] &= \int p(x)f(x)dx \\ &= \int \frac{q(x)}{q(x)}p(x)f(x)dx \\ &= \int q(x)\frac{p(x)}{q(x)}f(x)dx \\ &= E_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right] \end{aligned}$$

[IASI AI]

Importance sampling



$$\Delta Q(x,a) = \gamma^t \Big(\prod_{1 \le s \le t} \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)} \Big) \delta_t$$

Unbiased estimate of Q^π

Large (possibly infinite) variance

Not stable!

[Precup, Sutton, Singh, 2000], [Mahmood, Yu, White, Sutton, 2015] ,...

Why inf. variance? The correction factor is a product of many fractions that divides numbers smaller than 1 => a number too small or too big

[IASI AI]

General off-policy return-based algorithm:

$$\Delta Q(x,a) = \sum_{t \geq 0} \gamma^t \Big(\prod_{1 \leq s \leq t} c_s \Big) \underbrace{\big(r_t + \gamma \mathbb{E}_\pi Q(x_{t+1}, \cdot) - Q(x_t, a_t) \big)}_{\delta_t}$$

IS	$c_s = \frac{\pi(a_s \vert x_s)}{\mu(a_s \vert x_s)}$	high variance
$Q^\pi(\lambda)$	$c_s = \lambda$	not safe (off-policy)
$TB(\lambda)$	$c_s = \lambda \pi(a_s \vert x_s)$	not efficient (on-policy)
$Retrace(\lambda)$	$c_s = \lambda \min\left(1, \frac{\pi(a_s \vert x_s)}{\mu(a_s \vert x_s)}\right)$	none! (importance sampling truncated at 1)

Conclusion. Retrace(λ) can be seen as an algorithm that automatically adjusts – efficiently and safely – the length of the return to the degree of "off-policyness" of any available data

Safe and Efficient Off-Policy Reinforcement Learning

Upgoing policy update (UPGO) - last good experience > learned

Similar to self-imitation learning (exploit what the agent has just experienced is good, but has not yet learned) , the idea is to update the policy from partial trajectories with better-than-expected returns by bootstrapping (r + Gt) when the behaviour policy takes a worse-than-average action.

Policy – use V-Trace + Update the policy parameters in the direction of:

$$\rho_t(G_t^U - V_\theta(s_t, z))\nabla_\theta \log \pi_\theta(a_t \mid s_t, z)$$

Note:
The idea is:
Prefer to learn from a recent experience if gives better reward than your critic estimation

Q includes 1 current reward so it uses more of recent experience than just learned V estimation

Where:

$$G_t^U = \begin{cases} r_t + G_{t+1}^U & \text{if } Q(s_{t+1}, a_{t+1}, z) \geq V_\theta(s_{t+1}, z) \\ r_t + V_\theta(s_{t+1}, z) & \text{otherwise} \end{cases}$$

is an upgoing return
and $Q(s_t, a_t, z)$
is an action-value estimate

$$\rho_t = \min\left(\frac{\pi_\theta(a_t \mid s_t, z)}{\pi_{\theta'}(a_t \mid s_t, z)}, 1\right)$$

Note:
Use truncated importance sampling:
Critic should not trust too much in the actor decisions.
Also critic should not trust more than normal (let's say more than 1 to be like a probability) in his own decisions if actors says something very different.

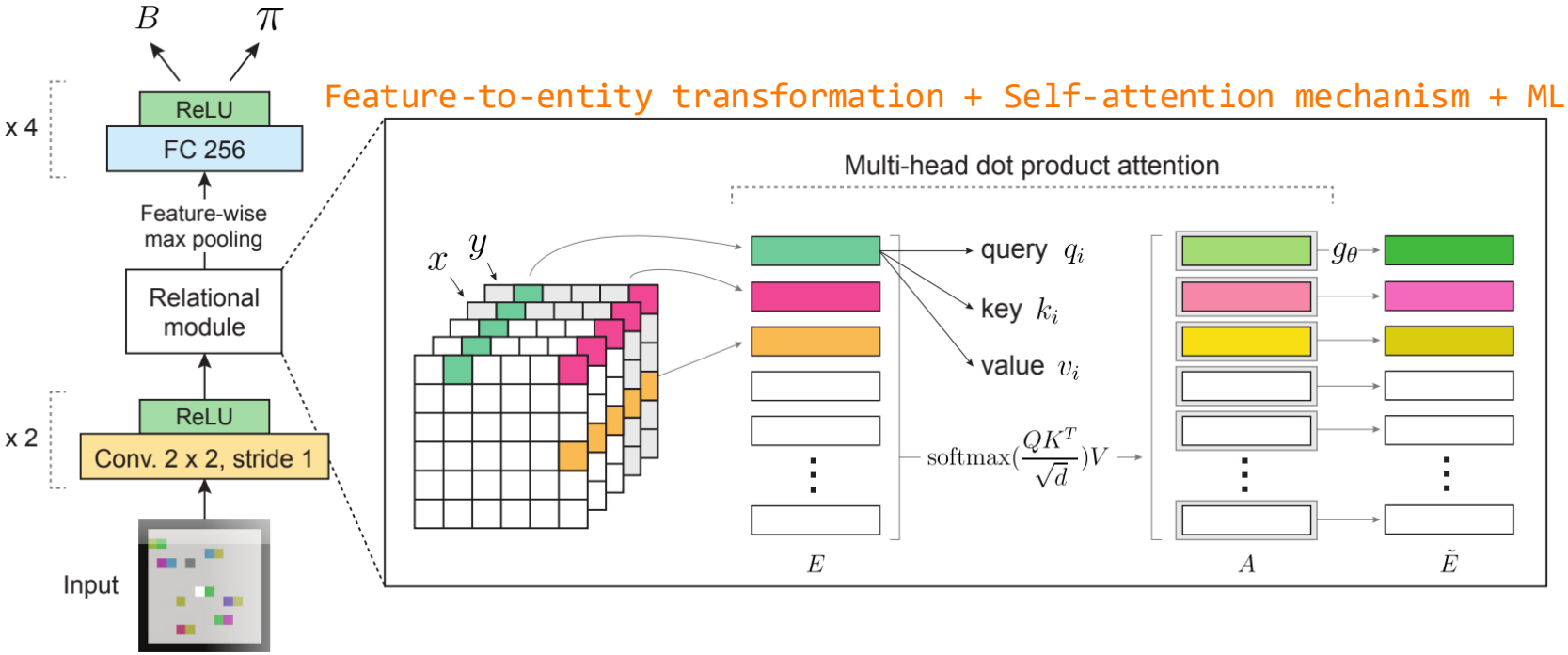
$\pi_{\theta'}$ policy that generated the trajectory in the actor.

Owing to the difficulty of approximating Q(st, at, z) over the large action space of StarCraft, we estimate action-values with a one-step target:

$$Q(s_t, a_t, z) = r_t + V_\theta(s_{t+1}, z)$$

[IASI AI]

Relational reasoning over structured representations



Box-World agent architecture and multi-head dot-product attention. E is a matrix that compiles the entities produced by the visual front-end; gθ is a multilayer perceptron applied in parallel to each row of the output of an MHDPA step, A, and producing updated entities, Ee.

…The output vector contains c-dimensional vector of π's logits where c is the number of discrete actions, plus a scalar baseline value estimate B

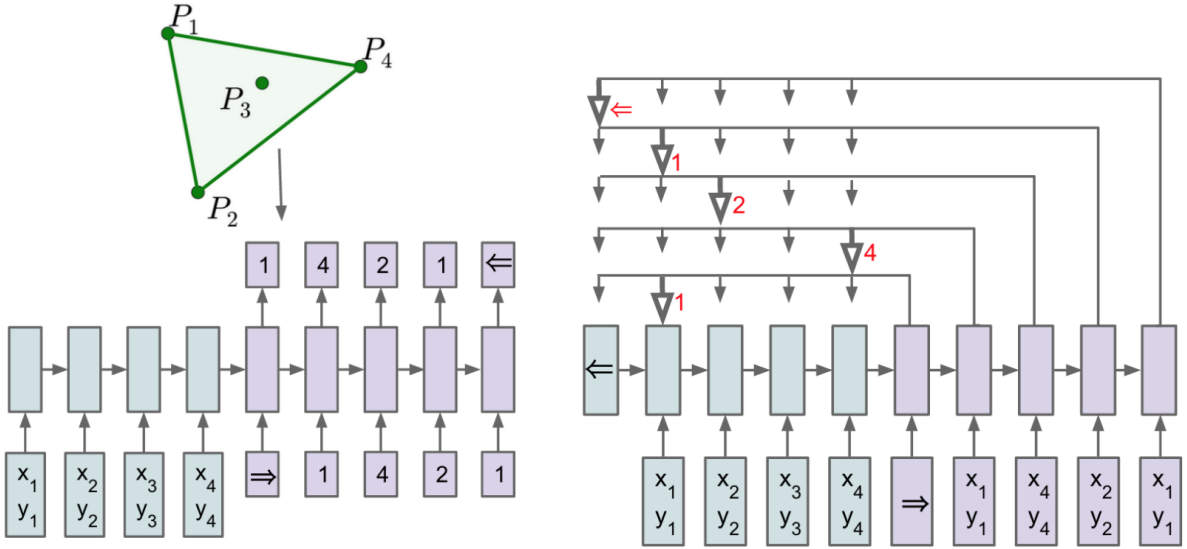
Deep reinforcement learning with relational inductive biases (Paper)

[IASI AI]

Pointer network – encoder-decoder with content-based attention mechanisms

Pointer Net is used to output the action for each unit, since the StarCraft involves many units in concert and the number of units changes over time.

Deciphering AlphaStar on StarCraft II



(a) Sequence-to-Sequence **(b) Ptr-Net**

(a) Sequence-to-Sequence - An RNN (blue) processes the input sequence to create a code vector that is used to generate the output sequence (purple) using the probability chain rule and another RNN. The output dimensionality is fixed by the dimensionality of the problem and it is the same during training and inference.

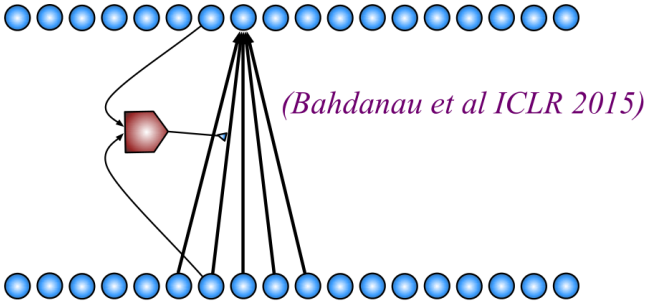
(b) Ptr-Net - An encoding RNN converts the input sequence to a code (blue) that is fed to the generating network (purple). At each step, the generating network produces a vector that modulates a content-based attention mechanism over inputs ([5, 2]). The output of the attention mechanism is a softmax distribution with dictionary size equal to the length of the input.

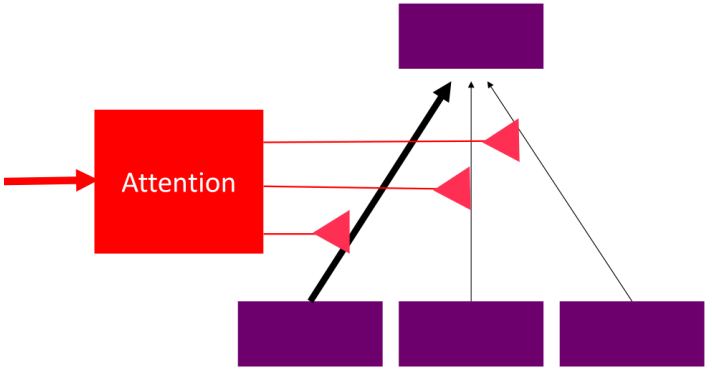
Paper

[IASI AI]

Core ingredient for Consciousness: Attention

- **Focus** on a one or a few elements at a time
- **Content-based soft attention** is convenient, can backprop to *learn where to attend*
- Attention is an **internal action**, needs a **learned attention policy** *(Egger et al 2019)*
- Self-attention: SOTA in NLP (transformers)



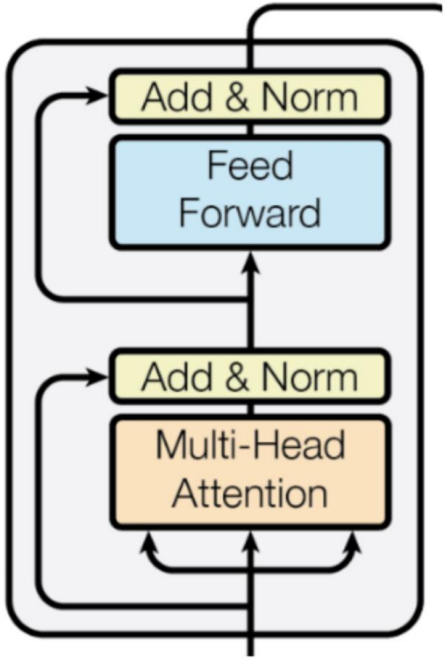


Attention = dynamic connection

Towards Compositional Understanding of the World by Agent-Based Deep Learning
From System 1 Deep Learning to System 2 Deep Learning

[IASI AI]

Attention Benefits



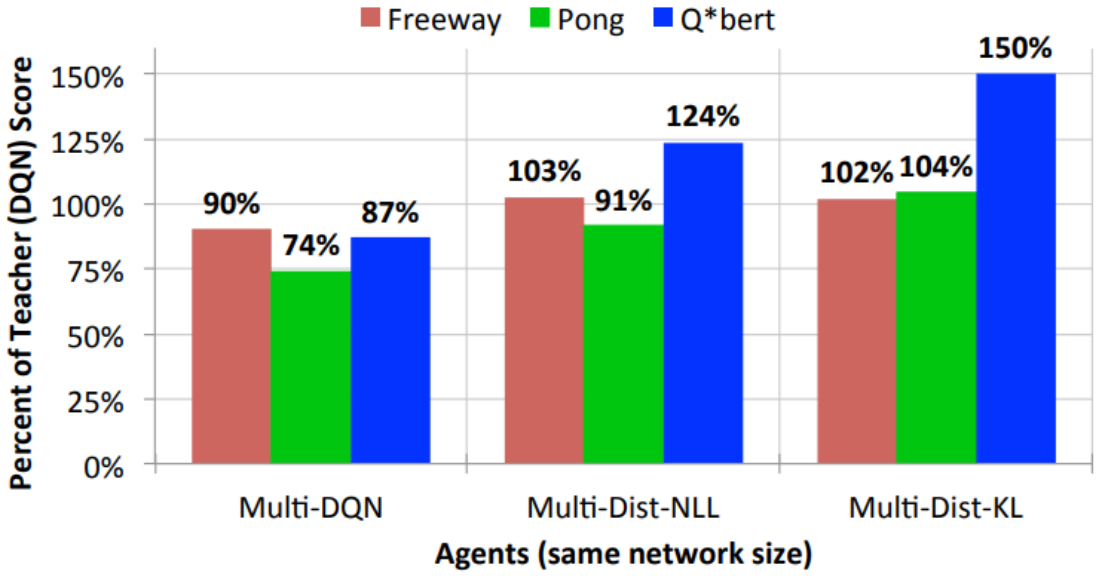
- Neural Machine Translation revolution
  (Bahdanau et al ICLR 2015)
- SOTA in NLP (self-attention, transformers)
- Memory-extended neural nets
- Address vanishing gradients *(Ke & al NeurIPS 2018)*
- Operating on unordered SETS of (key, value) pairs

The Illustrated Transformer
Attention? Attention!, The Annotated Transformer

[IASI AI]

Alphastar AI – Policy distillation

MULTI-GAME POLICY DISTILLATION RESULTS:



We train a multi-task DQN agent using the standard DQN algorithm applied to interleaved experience from three games (Multi-DQN), and compare it against distilled agents (Multi-Dist) which were trained using targets from three different single-game DQN teachers (see Figure 4). All three agents are using an identical multi-controller architecture of comparable size to a single teacher network. About 90% of parameters are shared, with only 3 small MLP “controllers” on top which are task specific and allow for different action sets between different games.

Policy Distillation

[FORTECH.AI]

[IASI AI]

Reinforcement learning Overview

Alex Movila

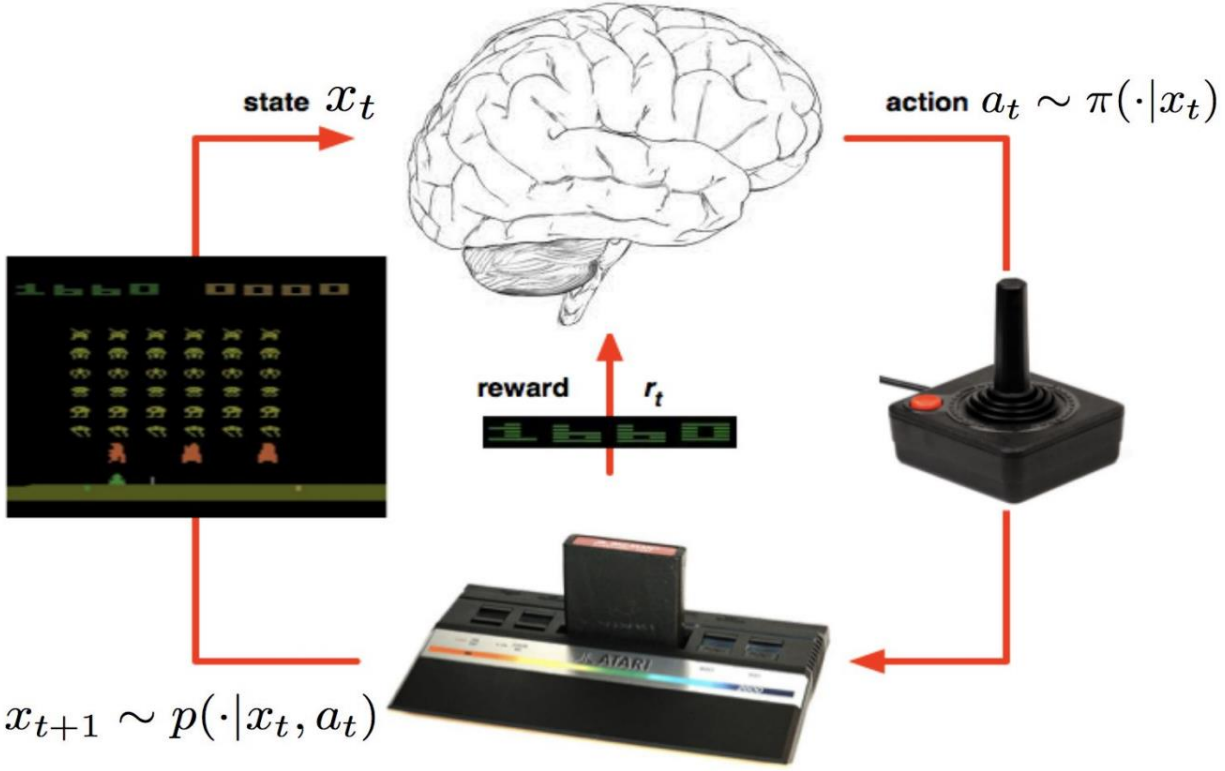
[FORTECH.AI] Introduction to Reinforcement Learning (RL)

- Learn to make good decisions
- No supervision. Learn from rewards

Two approaches:

- Value based ([Bellman, 1957]'s dynamic programming)
- Policy based ([Pontryagin, 1956]'s maximum principle)

[FORTECH.AI] Reinforcement Learning – Trial & Error learning based on rewards



How Does RL Relate to Other Machine Learning Problems?

Summary of differences between RL and supervised learning:

► You don't have full access to the function you're trying to optimize-must query it through interaction.

► Interacting with a *stateful world*: input Xt depend on your previous actions

[FORTECH.AI] Parameterized Policies

A family of policies indexed by parameter vector $\theta \in \mathbb{R}^d$

- Deterministic: $\pi(a|s,\theta)$
- Stochastic: $a = \pi(s,\theta)$

Analogous to classification or regression with input s, output a.

- Discrete action space:
  network outputs vector of probabilities
- Continuous action space:
  network outputs mean and diagonal covariance of Gaussian

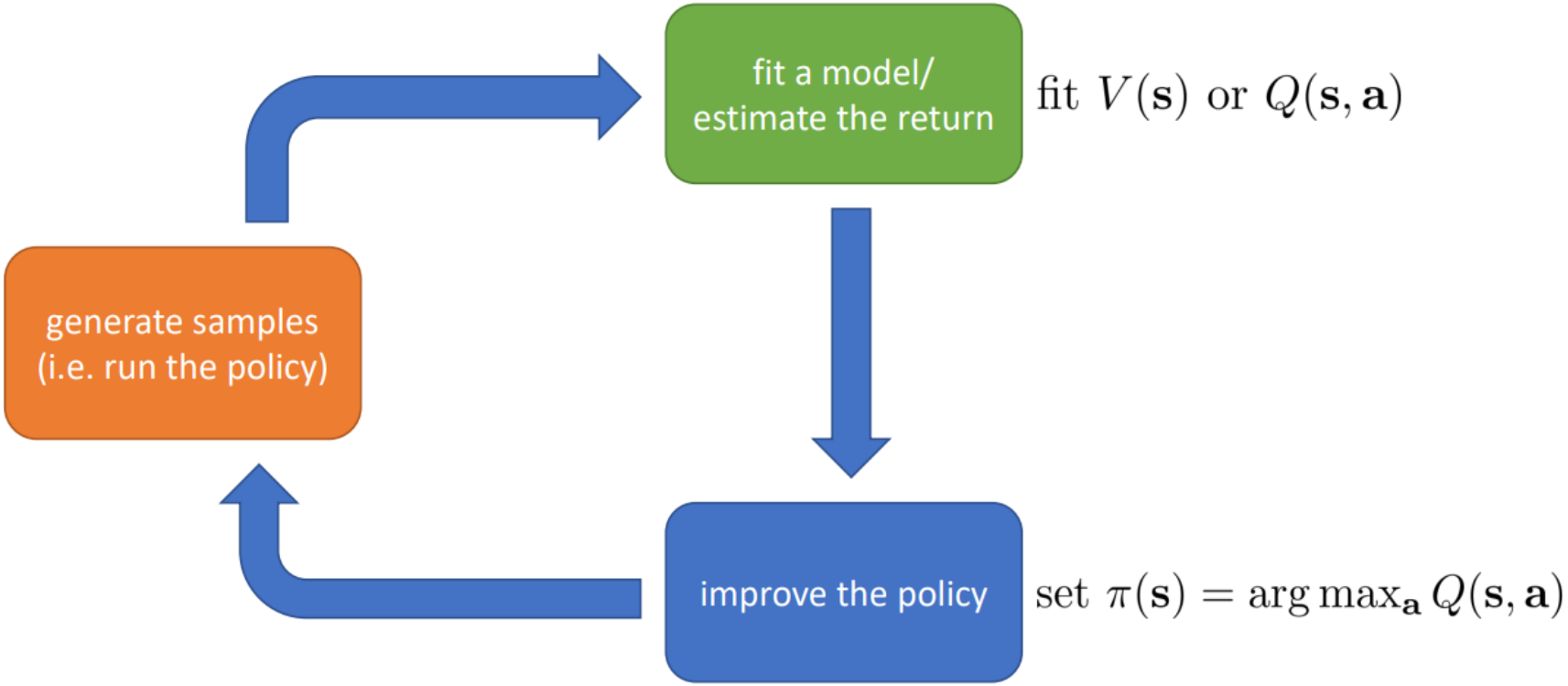
Types of RL algorithms

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

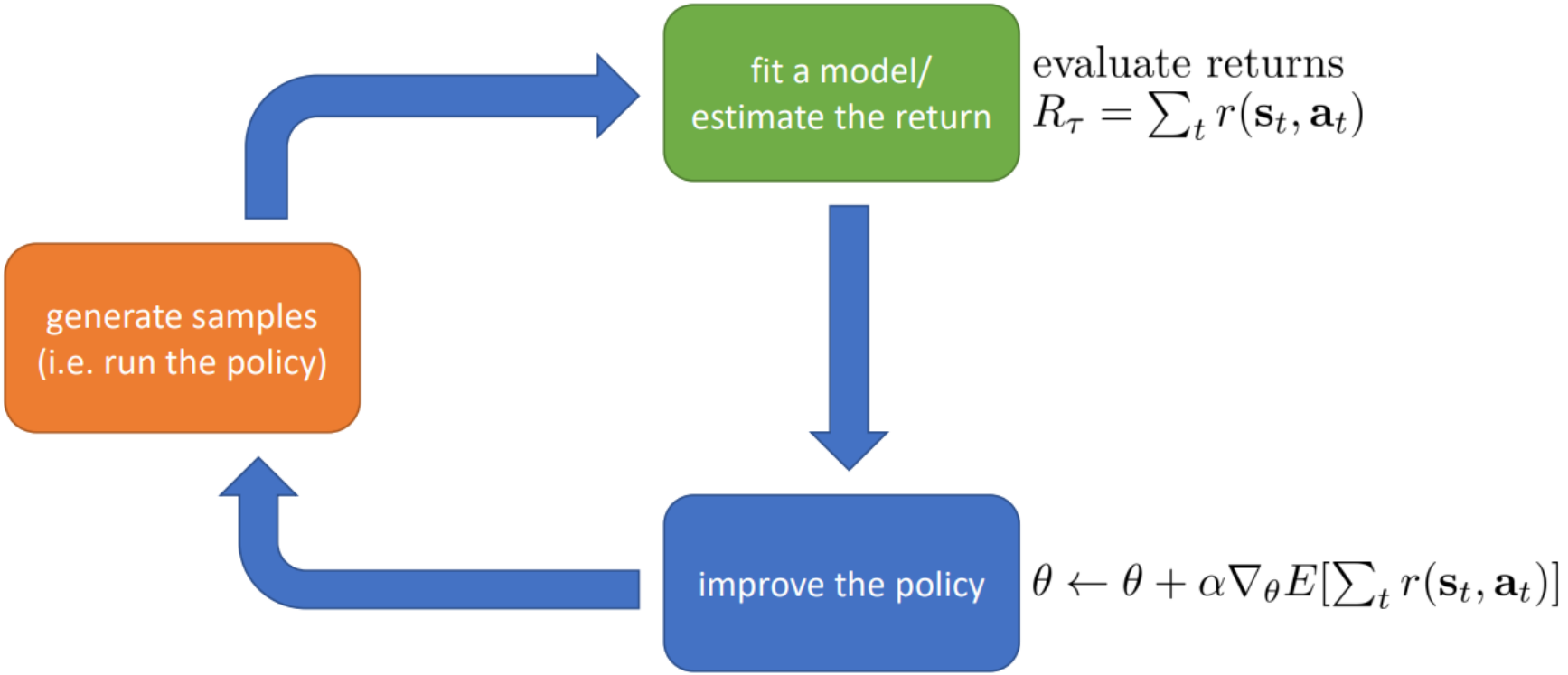
- Policy gradients: directly differentiate the above objective
- Value-based: estimate value function or Q-function of the optimal policy (no explicit policy)
- Actor-critic: estimate value function or Q-function of the current policy, use it to improve policy
- Model-based RL: estimate the transition model, and then…

- Use it for planning (no explicit policy)
- Use it to improve a policy
- Something else

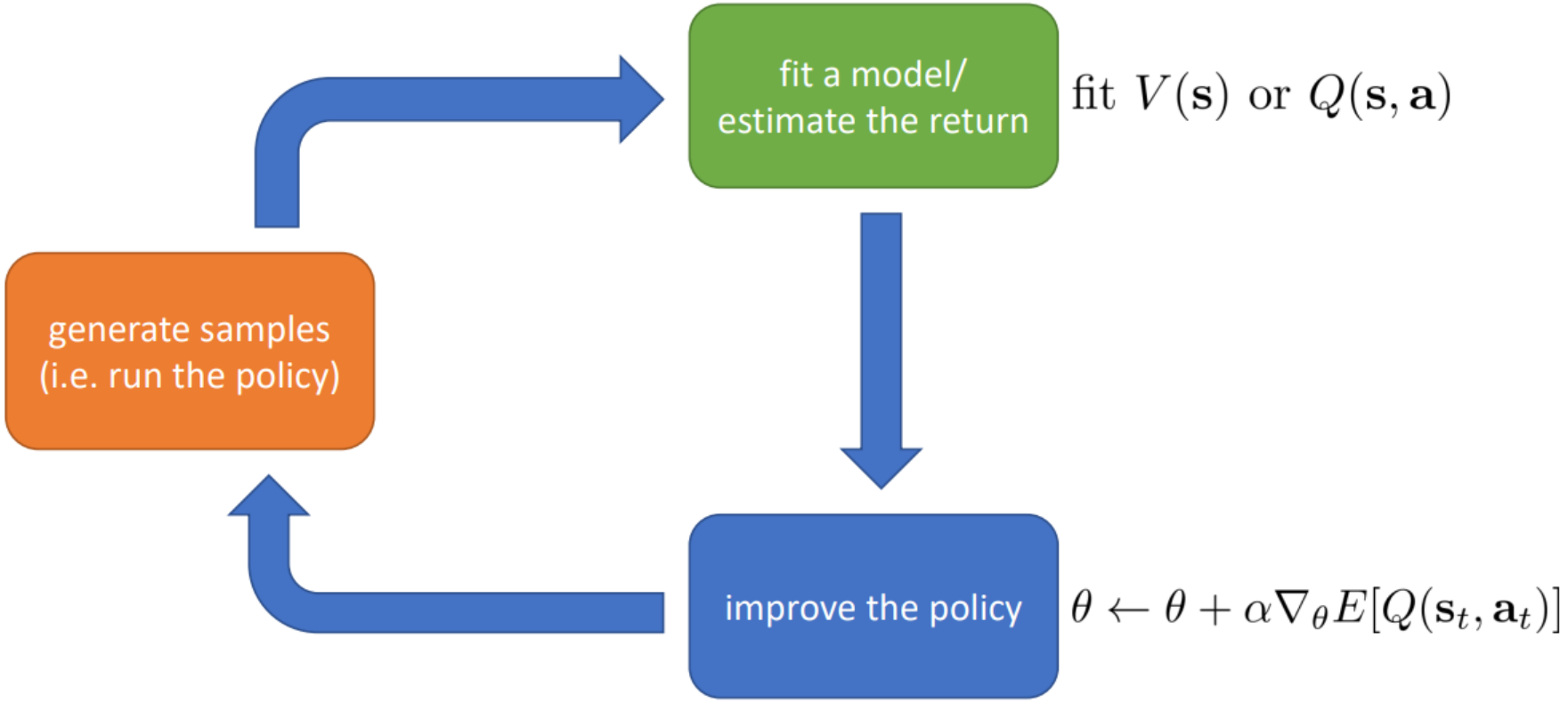
[FORTECH.AI] Value function based algorithms



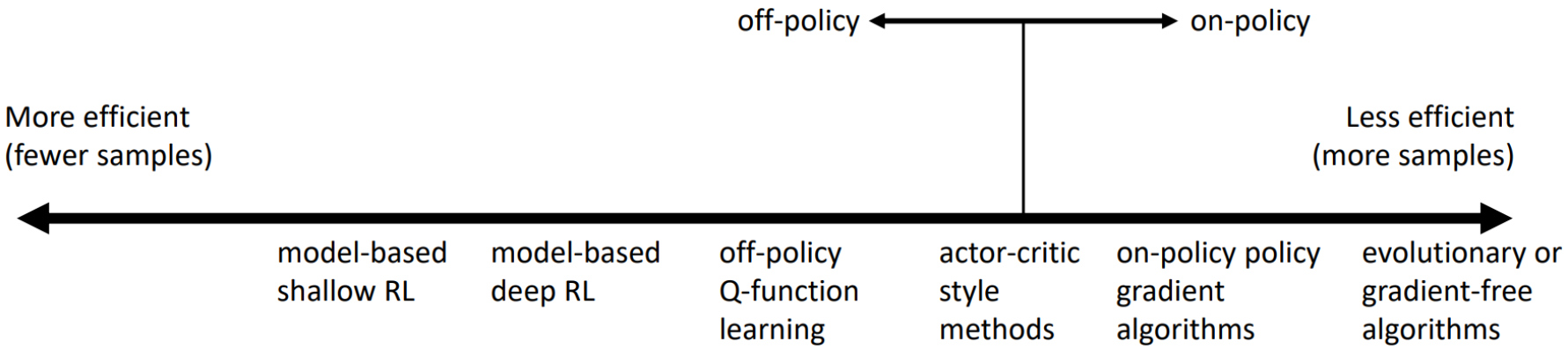
Direct policy gradients



Actor-critic: value functions + policy gradients



[FORTECH.AI] Comparison: sample efficiency



Why would we use a less efficient algorithm?

Wall clock time is not the same as efficiency!

[FORTECH.AI] Bellman's dynamic programming

- Define the value function Q^π of a policy $\pi(a|x)$:

$$Q^\pi(x,a) = \mathbb{E}\Big[\sum_{t\geq 0}\gamma^t r_t \Big| x, a, \pi\Big],$$

and the optimal value function:

$$Q^*(x,a) = \max_\pi Q^\pi(x,a).$$

(expected sum of future rewards if the agent plays optimally).

- Bellman equations:

$$Q^\pi(x,a) = r(x,a) + \gamma \mathbb{E}_{x'}\Big[\sum_{a'}\pi(a'|x')Q^\pi(x',a')\Big|x,a\Big]$$

$$Q^*(x,a) = r(x,a) + \gamma \mathbb{E}_{x'}\Big[\max_{a'} Q^*(x',a')\Big|x,a\Big]$$

- Optimal policy $\pi^*(x) = \arg\max_a Q^*(x,a)$

Expectation defined: "what you'd get if you were to sample x from probab. dist. p(x), and average the f(x) you got from those samples"

$$\mathbb{E}[f(x)] = \int_{x\sim p(x)} p(x) * f(x)$$

Q Learning - Represent Q using a neural network

- Represent value function $Q_w(x, a)$ with a neural net.
- How to train $Q_w(x, a)$? We don't have supervised values. We only know we want

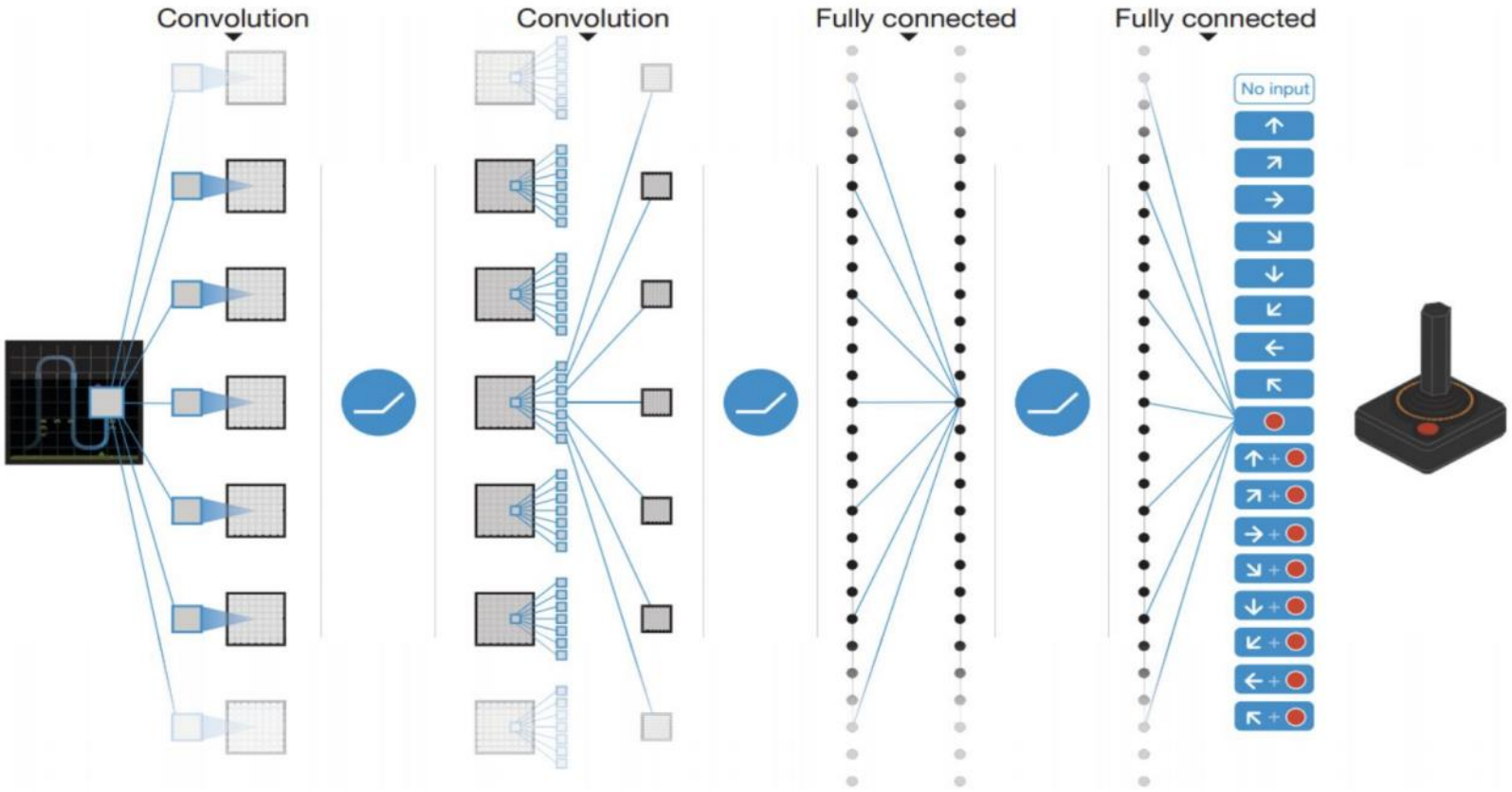
$$Q_w(x, a) \approx r(x, a) + \gamma \mathbb{E}_{x'} \left[\max_{a'} Q_w(x', a') \middle| x, a \right]$$

- After a transition $x_t, a_t \to x_{t+1}$,

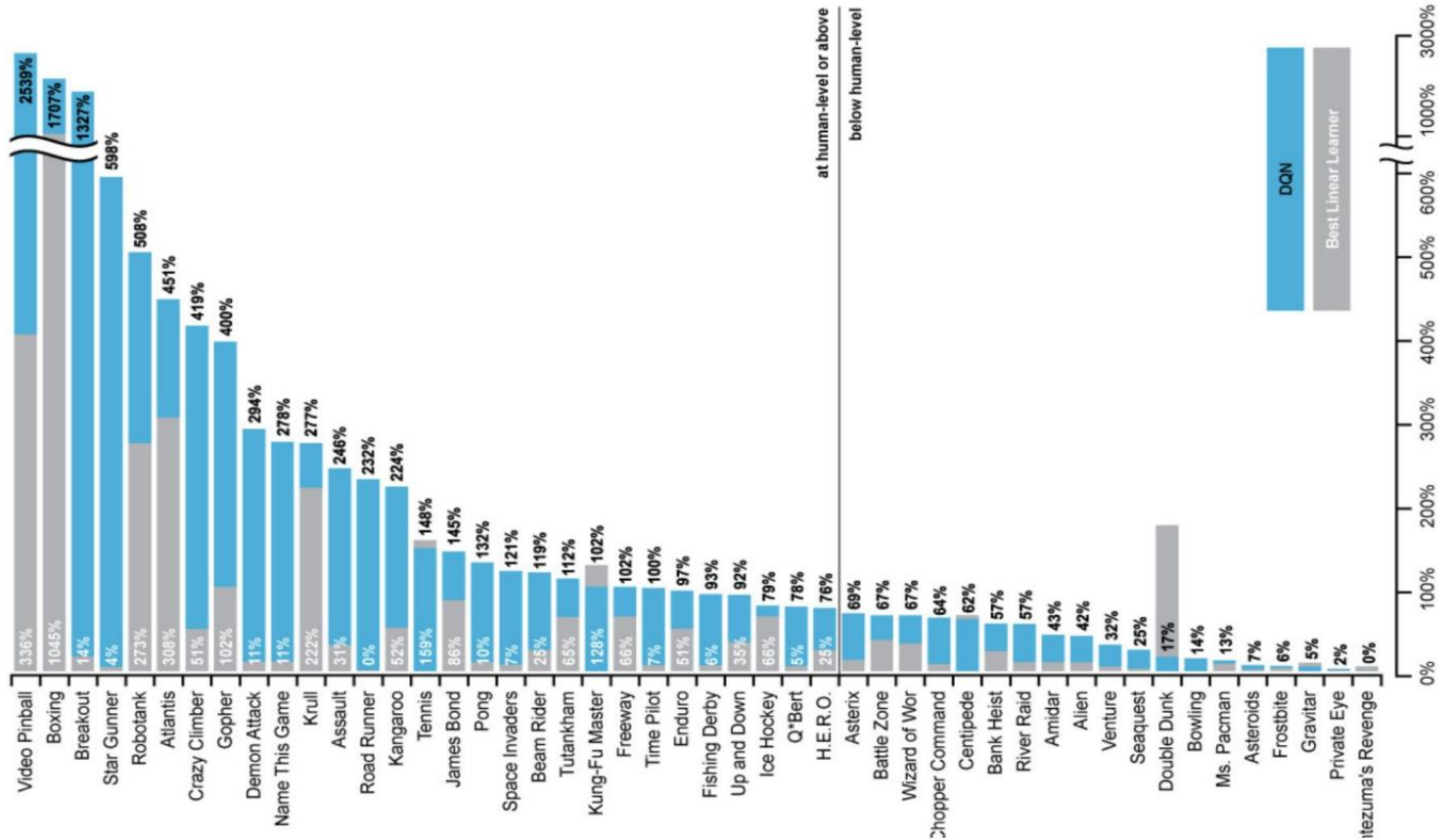
$$\text{train } Q_w(x_t, a_t) \text{ to predict } \underbrace{r_t + \gamma \max_a Q_w(x_{t+1}, a)}_{\text{target values}}$$

- Minimize loss $\Big(\underbrace{r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)}_{\text{temporal difference } \delta_t} \Big)^2$.
- At the end of learning, $\mathbb{E}[\delta_t] = 0$.

[FORTECH.AI] DQN network



[FORTECH.AI] DQN network – superhuman on most Atari games vs Best Linear Learner



Montezuma's Revenge – toughest game for DQN

[FORTECH.AI] Deep Q-Networks (DQN) [Mnih et al. 2013, 2015] - stabilize Q Learning

Problems: (1) data is not iid, (2) target values change

Idea: be as close as possible to supervised learning

1. Dissociate acting from learning:
   - Interact with the environments by following behavior policy
   - Store transition samples x_t, a_t, x_{t+1}, r_t into a memory replay
   - Train by replaying iid from memory
2. Use target network fixed for a while

$$\text{loss} = \left(r_t + \gamma \max_a Q_{w_{target}}(x_{t+1}, a) - Q_w(x_t, a_t) \right)^2$$

Properties: DQN is off-policy, and uses 1-step bootstrapping.

Note 1:
Q weights change during training

Note 2 - Separate policies is better:
Behaviour – conservative
Exploration for learning – Risky, non iid, crash

Note 3: The use of a replay memory to decorrelate samples is a critical element of DQN, as is the use of a target network, an older version of the parameters (θ − i). Both mechanisms help to stabilize learning.

Note 4:
...deep Q-learning and all of its variants, are they true off-policy algorithms? No, they are not. In Q-learning, we do something that's called epsilon-greedy exploration. Which means basically that with a small probability we choose a random action vs. the current optimal estimate of the action.

The False Promise of Off-Policy Reinforcement Learning Algorithms

Why policy optimization

- Often π can be simpler than Q or V
  - E.g., robotic grasp
- V: doesn't prescribe actions
  - Would need dynamics model (+ compute 1 Bellman back-up)
- Q: need to be able to efficiently solve $\arg\max_u Q_\theta(s, u)$
  - Challenge for continuous / high-dimensional action spaces*

* some recent work (parMally) addressing this:
NAF: Gu, Lillicrap, Sutskever, Levine ICML 2016
Input Convex NNs: Amos, Xu, Kolter arXiv 2016

- Parametrized policy $\pi_\theta(a|x)$
- **Policy gradient**: optimize

$$J(\theta) = \mathbb{E}\Big[\sum_{t\geq 0}\gamma^t r_t \Big| a_t \sim \pi_\theta(\cdot|x_t)\Big],$$

by gradient ascent:

$$\nabla J(\theta) = \mathbb{E}\Big[\sum_{t\geq 0}\gamma^t \nabla \log \pi_\theta(a_t|x_t)\big(\underbrace{r_t + \gamma r_{t+1} + \dots}_{Q^{\pi_\theta}(x_t,a_t)}\big)\Big]$$

Actor-critic algorithm learn both π_θ and Q_w.

Likelihood Ratio Policy Gradient

We let τ denote a state-action sequence $s_0, u_0, \ldots, s_H, u_H$. We overload notation: $R(\tau) = \sum_{t=0}^{H} R(s_t, u_t)$.

$$U(\theta) = \mathrm{E}[\sum_{t=0}^{H} R(s_t, u_t); \pi_\theta] = \sum_{\tau} P(\tau; \theta) R(\tau)$$

In our new notation, our goal is to find θ:

$$\max_\theta U(\theta) = \max_\theta \sum_{\tau} P(\tau; \theta) R(\tau)$$

Expectation defined:
"what you'd get if you were to sample x from probab. dist. p(x), and average the f(x) you got from those samples"

$$\mathbb{E}[f(x)] = \int_{x \sim p(x)} p(x) * f(x)$$

[FORTECH.AI] Likelihood Ratio Policy Gradient

$$U(\theta) = \sum_{\tau} P(\tau;\theta) R(\tau)$$

Taking the gradient w.r.t. θ gives

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_{\tau} P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau)
\end{aligned}$$

Approximate with the empirical estimate for m sample paths under policy π_θ:

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta) R(\tau^{(i)})$$

Expectation defined:
"what you'd get if you were to sample x from probab. dist. p(x), and average the f(x) you got from those samples"

$$\mathbb{E}[f(x)] = \int_{x \sim p(x)} p(x) * f(x)$$

- Sum and Difference Rule

$$h(x) = f(x) \pm g(x) \text{ then } h'(x) = f'(x) \pm g'(x)$$

- Chain Rule:

$$h(x) = f(g(x)) \text{ then } h'(x) = f'(g(x)) g'(x)$$

- Logarithm Derivatives

$$f(x) = \ln(x) \text{ then } f'(x) = \frac{1}{x}$$

$$f(x) = \ln(g(x)) \text{ then } f'(x) = \frac{g'(x)}{g(x)}$$

Deriv rules

Deep Reinforcement Learning Through Policy Optimization
CS 287 Advanced Robotics – Fundamental Knowledge

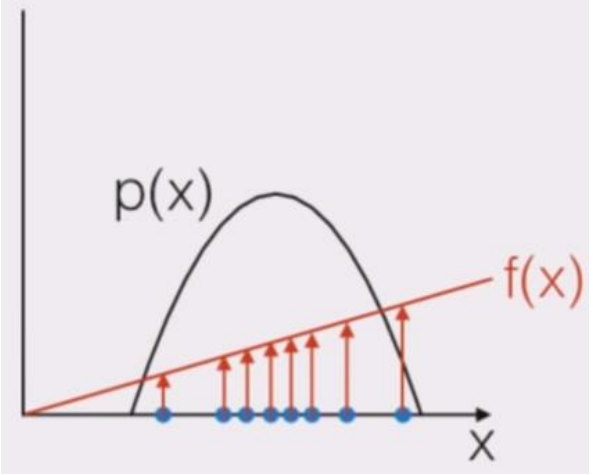
Likelihood Ratio Gradient: Validity

$$\nabla U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta) R(\tau^{(i)})$$

- Valid even if R is discontinuous, and unknown, or sample space (of paths) is a discrete set

[FORTECH.AI] Likelihood Ratio Gradient: Intuition

$$\hat{g}_i = f(x_i)\nabla_\theta \log p(x_i|\theta)$$



for each xi sample push probabilities up in proportion to f(xi)

- Let's say that f(x) measures how good the sample x is.
- Moving in the direction $\hat{g}_i$ pushes up the logprob of the sample, in proportion to how good it is
- Valid even if f(x) is discontinuous, and unknown, or sample space (containing x) is a discrete set

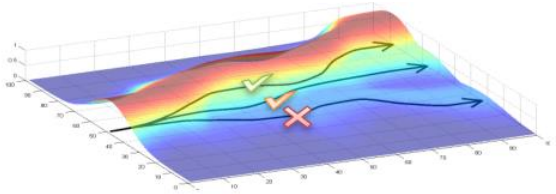
Deep Reinforcement Learning (John Schulman, OpenAI)

[FORTECH.AI] Likelihood Ratio Gradient: Intuition

$$\nabla U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta) R(\tau^{(i)})$$

Gradient tries to:

- Increase the probability of paths with positive R
- Decrease the probability of paths with negative R



! Likelihood ratio changes probabilities of experienced paths, does not try to change the paths (see Path Derivative later)

Policy gradient objective – dynamics not needed

Let's decompose the path into states and actions => Dynamics model is not required to compute the policy gradient

$$\nabla_\theta \log P(\tau^{(i)};\theta) = \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)}|s_t^{(i)},u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)}|s_t^{(i)})}_{\text{policy}} \right]$$

$$= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}^{(i)}|s_t^{(i)},u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) \right]$$

$$= \nabla_\theta \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)})$$

$$= \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})}_{\text{no dynamics model required!!}}$$

Logarithm product rule

$$\log_b(x \cdot y) = \log_b(x) + \log_b(y)$$

- **Constant Rule:**

$$f(x) = c \text{ then } f'(x) = 0$$

Likelihood Ratio Gradient Estimate

The following expression provides us with an unbiased estimate of the gradient, and we can compute it without access to a dynamics model:

$$\hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta) R(\tau^{(i)})$$

Here:

$$\nabla_\theta \log P(\tau^{(i)}; \theta) = \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{no dynamics model required!!}}$$

Unbiased means:

$$\mathrm{E}[\hat{g}] = \nabla_\theta U(\theta)$$

Given enough sampled trajectories we can estimate accurately the gradient of expected total reward.

[FORTECH.AI] Likelihood Ratio Gradient Estimate – High variance problem

As formulated thus far: gradient estimate is unbiased (good direction) but very noisy

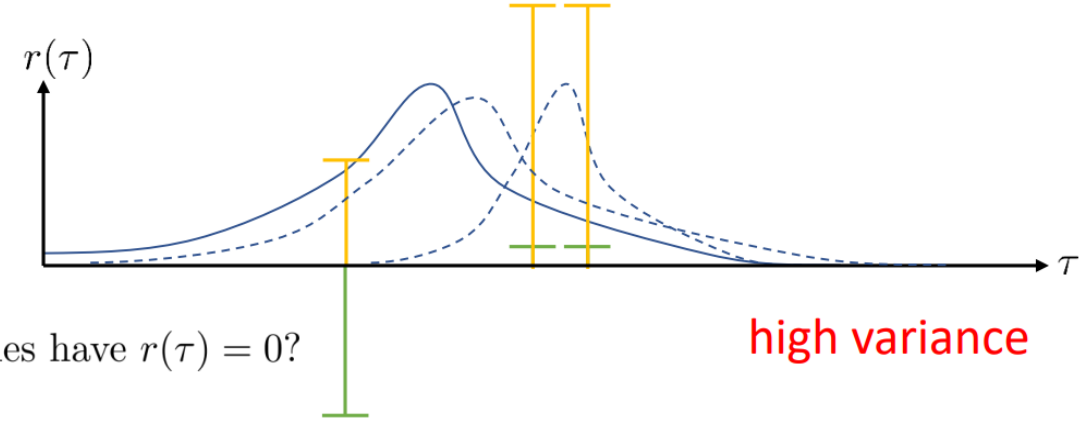
Fixes that lead to real-world practicality
- Baseline
- Temporal structure

- Also: KL-divergence trust region / natural gradient (= general trick, equally applicable to perturbation analysis and finite differences)

[FORTECH.AI] What is wrong with the policy gradient?

Just adding a constant to sampled rewards changes the optimal estimated policy.

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau) r(\tau)$$



even worse: what if the two "good" samples have $r(\tau) = 0$?

high variance

Policy gradient is on-policy

$$\theta^\star = \arg\max_\theta J(\theta)$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

this is trouble...

- Neural networks change only a little bit with each gradient step
- On-policy learning can be extremely inefficient!

can't just skip this!

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

Off-policy learning & importance sampling

$$\theta^\star = \arg\max_\theta J(\theta)$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

what if we don't have samples from $\pi_\theta(\tau)$?

(we have samples from some $\bar{\pi}(\tau)$ instead)

$$J(\theta) = E_{\tau\sim\bar{\pi}(\tau)}\left[\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)} r(\tau)\right]$$

$$\pi_\theta(\tau) = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$$

$$\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)} = \frac{\cancel{p(\mathbf{s}_1)}\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}}{\cancel{p(\mathbf{s}_1)}\prod_{t=1}^{T}\bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)\cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}} = \frac{\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^{T}\bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)}$$

importance sampling

$$\begin{aligned} E_{x\sim p(x)}[f(x)] &= \int p(x)f(x)dx \\ &= \int \frac{q(x)}{q(x)}p(x)f(x)dx \\ &= \int q(x)\frac{p(x)}{q(x)}f(x)dx \\ &= E_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right] \end{aligned}$$

The off-policy policy gradient

$$\theta^\star = \arg\max_\theta J(\theta)$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

$$\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)} = \frac{\prod_{t=1}^T \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^T \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}$$

$$\nabla_{\theta'} J(\theta') = E_{\tau\sim\pi_\theta(\tau)}\left[\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)}\nabla_{\theta'}\log\pi_{\theta'}(\tau)r(\tau)\right] \quad \text{when } \theta \neq \theta'$$

$$= E_{\tau\sim\pi_\theta(\tau)}\left[\left(\prod_{t=1}^T \frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}\right)\left(\sum_{t=1}^T \nabla_{\theta'}\log\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^T r(\mathbf{s}_t,\mathbf{a}_t)\right)\right] \quad \text{what about causality?}$$

$$= E_{\tau\sim\pi_\theta(\tau)}\left[\sum_{t=1}^T \nabla_{\theta'}\log\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)\left(\prod_{t'=1}^t \frac{\pi_{\theta'}(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{\pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})}\right)\left(\sum_{t'=t}^T r(\mathbf{s}_{t'},\mathbf{a}_{t'})\left(\prod_{t''=t}^{t'} \frac{\pi_{\theta'}(\mathbf{a}_{t''}|\mathbf{s}_{t''})}{\pi_\theta(\mathbf{a}_{t''}|\mathbf{s}_{t''})}\right)\right)\right]$$

future actions don't affect current weight

if we ignore this, we get
a policy iteration algorithm
(more on this in a later lecture)

[FORTECH.AI] Critics as state-dependent baselines

Actor-critic: $$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\left(r(\mathbf{s}_{i,t},\mathbf{a}_{i,t}) + \gamma \hat{V}^\pi_\phi(\mathbf{s}_{i,t+1}) - \hat{V}^\pi_\phi(\mathbf{s}_{i,t})\right)$$

+ lower variance (due to critic)

- not unbiased (if the critic is not perfect)

Policy gradient: $$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\left(\left(\sum_{t'=t}^{T}\gamma^{t'-t} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right) - b\right)$$

+ no bias

- higher variance (because single-sample estimate)

can we use $\hat{V}^\pi_\phi$ and still keep the estimator unbiased?

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\left(\left(\sum_{t'=t}^{T}\gamma^{t'-t} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right) - \hat{V}^\pi_\phi(\mathbf{s}_{i,t})\right)$$

+ no bias

+ lower variance (baseline is closer to rewards)

Eligibility traces & n-step returns

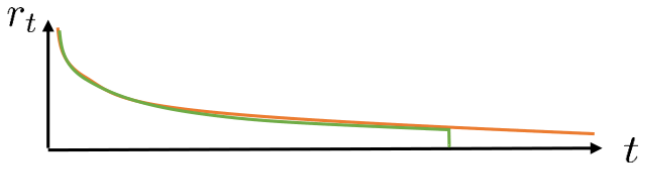
$$\hat{A}^\pi_{\mathrm{C}}(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \hat{V}^\pi_\phi(\mathbf{s}_{t+1}) - \hat{V}^\pi_\phi(\mathbf{s}_t)$$

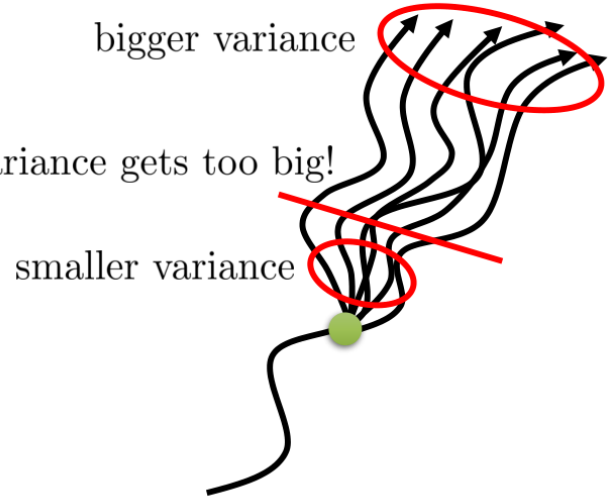
+ lower variance
- higher bias if value is wrong (it always is)

$$\hat{A}^\pi_{\mathrm{MC}}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}^\pi_\phi(\mathbf{s}_t)$$

+ no bias
- higher variance (because single-sample estimate)

Can we combine these two, to control bias/variance tradeoff?

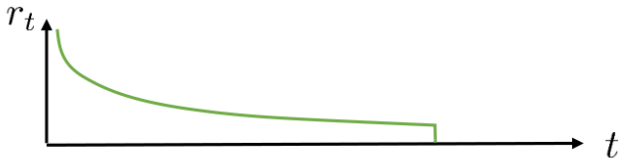




$$\hat{A}^\pi_n(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}^\pi_\phi(\mathbf{s}_t) + \gamma^n \hat{V}^\pi_\phi(\mathbf{s}_{t+n})$$

choosing $n > 1$ often works better!

Generalized advantage estimation



Do we have to choose just one n?

Cut everywhere all at once!

$$\hat{A}_n^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - \hat{V}_\phi^\pi(\mathbf{s}_t) + \gamma^n \hat{V}_\phi^\pi(\mathbf{s}_{t+n})$$

$$\hat{A}_{\text{GAE}}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{n=1}^{\infty} w_n \hat{A}_n^\pi(\mathbf{s}_t, \mathbf{a}_t)$$ Weighted combination of n-step returns

How to weight? Mostly prefer cutting earlier (less variance) $w_n \propto \lambda^{n-1}$ exponential falloff

$$\hat{A}_{\text{GAE}}^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma((1-\lambda)\hat{V}_\phi^\pi(\mathbf{s}_{t+1}) + \lambda(r(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) + \gamma((1-\lambda)\hat{V}_\phi^\pi(\mathbf{s}_{t+2}) + \lambda r(\mathbf{s}_{t+2}, \mathbf{a}_{t+2}) + \dots)$$

$$\hat{A}_{\text{GAE}}^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} (\gamma\lambda)^{t'-t} \delta_{t'} \qquad \delta_{t'} = r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{t'+1}) - \hat{V}_\phi^\pi(\mathbf{s}_{t'})$$

similar effect as discount!

option 1: $$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

remember this?
discount = variance reduction!

Schulman, Moritz, Levine, Jordan, Abbeel '16

Likelihood Ratio Gradient Estimate : Baseline

$$\nabla U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta) R(\tau^{(i)})$$

- To build intuition, let's assume R > 0
  - Then tries to increase probabilities of all paths

→ Consider baseline b:

$$\nabla U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta)(R(\tau^{(i)}) - b)$$

still unbiased
[Williams 1992]

$$\begin{aligned}
&\mathbb{E}\left[\nabla_\theta \log P(\tau;\theta) b\right] \\
&= \sum_\tau P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) b \\
&= \sum_\tau P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} b \\
&= \sum_\tau \nabla_\theta P(\tau;\theta) b \\
&= \nabla_\theta \left(\sum_\tau P(\tau) b \right) \\
&= \nabla_\theta (b) \\
&= 0
\end{aligned}$$

Good choices for b?

$$b = \mathbb{E}\left[R(\tau)\right] \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau^{(i)})$$

$$b = \frac{\sum_i \left(\nabla_\theta \log P(\tau^{(i)};\theta)\right)^2 R(\tau^{(i)})}{\sum_i \left(\nabla_\theta \log P(\tau^{(i)};\theta)\right)^2}$$

[See: Greensmith, Bartlett, Baxter, JMLR 2004 for variance reduction techniques.]

Likelihood Ratio and Temporal Structure

- Current estimate:

$$\hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta)(R(\tau^{(i)}) - b)$$

$$= \frac{1}{m}\sum_{i=1}^{m} \left(\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\right)\left(\sum_{t=0}^{H-1} R(s_t^{(i)}, u_t^{(i)}) - b\right)$$

- Future actions do not depend on past rewards, hence can lower variance by instead using:

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\left(\sum_{k=t}^{H-1} R(s_k^{(i)}, u_k^{(i)}) - b(s_k^{(i)})\right)$$

- Good choice for b?

Expected return: $b(s_t) = \mathbb{E}\left[r_t + r_{t+1} + r_{t+2} + \ldots + r_{H-1}\right]$

→ Increase logprob of action proportionally to how much its returns are better than the expected return under the current policy

[FORTECH.AI] Baselines in Policy gradient for variance reduction

Expected Grad-Log-Prob Lemma

EGLP Lemma. Suppose that is a parameterized probability distribution over a random variable,.
Then:

$$\mathop{\mathrm{E}}_{x\sim P_\theta}[\nabla_\theta \log P_\theta(x)] = 0.$$

The demonstration uses:
- integral of a distrib (area under distrib plot) = 1
- log trick

Baselines in Policy Gradients

An immediate consequence of the EGLP lemma is that for any function which only depends on state,

$$\mathop{\mathrm{E}}_{a_t\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t)b(s_t)] = 0.$$

<= deriv of log product rule or
<= • Constant Multiple Rule $g(x) = c \cdot f(x)$ then $g'(x) = c \cdot f'(x)$

This allows us to add or subtract any number of terms like this from our expression for the policy gradient, without changing it in expectation:
Vanilla policy gradient expression with value function baseline:

$$\nabla_\theta U(\theta) \approx \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H}\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\left(\sum_{k=t}^{H-1} R(s_k^{(i)}, u_k^{(i)}) - V_\phi(s_t)\right)$$

Note: term: log pi * V(St) = 0

[FORTECH.AI] Other forms of policy gradient

Other Forms of the Policy Gradient

So policy gradient has the general form

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\Phi_t\right]$$

$$\Phi_t = R(\tau),$$

Reward-to-Go Policy Gradient (Don't Let the Past Distract You) **- we sum only upcoming rewards**

$$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}),$$

$$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t).$$

$$\Phi_t = Q^{\pi_\theta}(s_t, a_t) \qquad \Phi_t = A^{\pi_\theta}(s_t, a_t)$$

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

, describes how much better or worse it is than other actions on average (relative to the current policy)

How to learn the value estimator / baseline network:
The simplest method for learning the baseline, used in most implementations of policy optimization algorithms (including PPO, and A2C), is to minimize a mean-squared-error objective:

$$\phi_k = \arg\min_\phi \underset{s_t, \hat{R}_t\sim\pi_k}{\mathrm{E}}\left[\left(V_\phi(s_t) - \hat{R}_t\right)^2\right]$$

The Delayed Reward Problem

- With policy gradient methods, we are confounding the effect of multiple actions:

$$\hat{A}_t = r_t + r_{t+1} + r_{t+2} + \cdots - b(s_t)$$

  mixes effect of $a_t, a_{t+1}, a_{t+2}, \ldots$
- SNR of $\hat{A}_t$ scales roughly as $1/T$
  - Only a_t contributes to *signal* $A^\pi(s_t, a_t)$, but $a_{t+1}, a_{t+2}, \ldots$ contribute to noise.

[FORTECH.AI] Variance Reduction with Discounts

- Introduce discount fac γ , which ignores delayed effects between actions and rewards

$$\nabla_\theta \mathbb{E}_\tau[R] \approx \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t,\theta)\left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} - b(s_t)\right)\right]$$

- Now we want:

$$b(s_t) \approx \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-1-t} r_{T-1}\right]$$

[FORTECH.AI] Variance Reduction with Discounts

- Discount factor γ, $0 < \gamma < 1$, downweights the effect of rewars that are far in the future—ignore long term dependencies
- We can form an advantage estimator using the *discounted return*:

$$\hat{A}_t^{\gamma} = \underbrace{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots}_{\text{discounted return}} - b(s_t)$$

  reduces to our previous estimator when $\gamma = 1$.
- So advantage has expectation zero, we should fit baseline to be *discounted value function*

$$V^{\pi,\gamma}(s) = \mathbb{E}_{\tau}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \dots \mid s_0 = s\right]$$

- Discount γ is similar to using a horizon of $1/(1-\gamma)$ timesteps
- $\hat{A}_t^{\gamma}$ is a biased estimator of the advantage function

Value Functions in the Future

- Baseline accounts for and removes the effect of *past* actions
- Can also use the value function to estimate future rewards

$$r_t + \gamma V(s_{t+1}) \qquad \text{cut off at one timestep}$$

$$r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) \qquad \text{cut off at two timesteps}$$

$$\dots$$

$$r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \qquad \infty \text{ timesteps (no } V)$$

Value Functions in the Future

- Subtracting out baselines, we get advantage estimators

$$\hat{A}_t^{(1)} = r_t + \gamma V(s_{t+1}) - V(s_t)$$
$$\hat{A}_t^{(2)} = r_t + r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t)$$
$$\dots$$
$$\hat{A}_t^{(\infty)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots - V(s_t)$$

- $\hat{A}_t^{(1)}$ has low variance but high bias, $\hat{A}_t^{(\infty)}$ has high variance but low bias.
- Using intermediate k (say, 20) gives an intermediate amount of bias and variance

Further Variance Reduction - Finite-Horizon Methods: Advantage Actor-Critic

Previously discussed:
- use rewards to go
- use discounted future rewards
- use baselines

As baseline use value functions for more variance reduction (at the cost of bias): actor-critic methods

A2C / A3C uses this fixed-horizon advantage estimator

Pseudo-code Reinforce aka Vanilla Policy Gradient

Algorithm 1 "Vanilla" policy gradient algorithm

Initialize policy parameter θ, baseline b

for iteration=$1, 2, \ldots$ **do**

Collect a set of trajectories by executing the current policy

At each timestep in each trajectory, compute

the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and

the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.

Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,

summed over all trajectories and timesteps.

Update the policy, using a policy gradient estimate $\hat{g}$,

which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta)\hat{A}_t$

end for

Deep Reinforcement Learning through Policy Optimization (John Schulman, OpenAI) (Talk) (Slides)

[FORTECH.AI] Step size matters in RL – behaviour change leads to totally new data

Why are step sizes a big deal in RL?

- Supervised learning
  - Step too far → next updates will fix it
- Reinforcement learning
  - Step too far → bad policy
  - Next batch: collected under bad policy
  - Can't recover, collapse in performance!

- due to forgetting what you learned from old data

One approach – slowly change policy:
Trust Region Policy Optimization (TRPO) :
limit KL divergencebetween action distribution of pre-update and post-update policy

$$\mathbb{E}_s\left[D^{\mathrm{KL}}(\pi_{\mathrm{old}}(\cdot|s)\|\pi(\cdot|s))\right] \leq \delta$$

[FORTECH.AI]

[IASI AI]

Deep RL – Links

Alex Movila

[FORTECH.AI] Other Links

Frontier algorithm ideas for deep reinforcement learning
Understanding Evolved Policy Gradients
Curiosity in Deep Reinforcement Learning
WHAT ARE MAJOR REINFORCEMENT LEARNING ACHIEVEMENTS & PAPERS FROM 2018?
BREAKTHROUGH RESEARCH IN REINFORCEMENT LEARNING FROM 2019
Off-Policy Actor-Critic with Shared Experience Replay
Dopamine and temporal difference learning: A fruitful relationship between neuroscience and AI
What's New in Deep Learning Research: How DeepMind Builds Multi-Task Reinforcement Learning Algorithms That Don't Get Distracted
The AI Algorithm That Mimics Our Brain's Dopamine Reward System
DeepMind's MuZero teaches itself how to win at Atari, chess, shogi, and Go
What's hot in AI: Deep reinforcement learning
The False Promise of Off-Policy Reinforcement Learning Algorithms
Deep reinforcement learning with relational inductive biases
Reinforcement Learning Papers Accepted to ICLR 2020
My Top 10 Deep RL Papers of 2019
Dopamine and temporal difference learning: A fruitful relationship between neuroscience and AI
Optimistic Actor Critic avoids the pitfalls of greedy exploration in reinforcement learning

[FORTECH.AI] Links – Tutorial DRL

Open AI - Spinning Up in Deep RL
Lil'Log blog - RL overview
An Outsider's Tour of Reinforcement Learning
A Brief Survey of Deep Reinforcement Learning
A Beginner's Guide to Deep Reinforcement Learning
Jeremy Zhang Blog
RLlib: Scalable Reinforcement Learning
Open-Source Libraries Cheat Sheet
RL Alg Cheat Sheet
RL Seminar
Deep Reinforcement Learning
An Introduction to Deep Reinforcement Learning
Understanding Reinforcement Learning Math, for Developers
Deep Reinforcement Learning Series

[FORTECH.AI] Links – Tutorial DRL

Why Going from Implementing Q-learning to Deep Q-learning Can Be Difficult
Proximal Policy Optimization Tutorial
The Pursuit of (Robotic) Happiness: How TRPO and PPO Stabilize Policy Gradient Methods
Deep Q Network vs Policy Gradients - An Experiment on VizDoom with Keras
Deep Reinforcement Learning in Python Tutorial - A Course on How to Implement Deep Learning Papers

[FORTECH.AI] Links – Imitation Learning

RL — Imitation Learning
Deep Q-learning from Demonstrations
Combining Imitation Learning and Reinforcement Learning Using DQfD
Deep Q-learning from Demonstrations (DQfD) in Keras

[FORTECH.AI] Links – Tutorial DRL – Presentations & Talks

Deep Reinforcement Learning through Policy Optimization (John Schulman, OpenAI) (Talk) (Slides)
Deep Reinforcement Learning Through Policy Optimization (NIPS2016)

off-policy deep RL , Remi Munos, Deepmind Paris (PPT)
Reinforcement Learning: Past, Present, and Future Perspectives (Slides)
Policy Gradient methods and Proximal Policy Optimization (PPO): diving into Deep RL!
Deep Reinforcement Learning: from the basics to recent algorithms - Olivier Sigaud (Slides)
Policy Gradient Methods: Tutorial and New Frontiers
OpenAI - Spinning Up in Deep RL Workshop (Slides and Materials) (RL Intro)
Reinforcement Learning: Past, Present, and Future Perspectives (slides)
Deep Reinforcement Learning in the Real World -Sergey Levine
NeurIPS 2019 - Deep Reinforcement Learning track

[FORTECH.AI] Links – Courses DRL

Deep RL Bootcamp (Berkeley 2017)
CS 285 at UC Berkeley - Deep Reinforcement Learning
CS 287: Advanced Robotics, Fall 2019
Stanford CS234: Reinforcement Learning | Winter 2019
NUS CS 6101 - Deep Reinforcement Learning
RL Course by David Silver
Advanced Deep Learning & Reinforcement Learning
AI - Advanced Deep Learning Course
Deep RL Bootcamp Frontiers
Reinforcement Learning - Introducing Goal Oriented Intelligence
Reinforcement Learning (sentdex)
RL class - Olivier Sigaud
Workshop Deep Reinforcement Learning
OpenAI - Spinning Up in Deep RL Workshop
Policy Gradient Methods: Tutorial and New Frontiers
Free Reinforcement Learning Course
Reinforcement Learning in the Open AI Gym
Deep Reinforcement Learning Tutorials
Workshop on New Directions in Reinforcement Learning and Control
The Reinforcement Learning Specialization

[FORTECH.AI] Links – DRL – Limits

Deep Reinforcement Learning Doesn't Work Yet
Challenges of Real-World Reinforcement Learning
Deep Reinforcement Learning is a waste of time
Why Reinforcement Learning is Wrong for Your Business
The False Promise of Off-Policy Reinforcement Learning Algorithms

Implementation Matters in Deep RL: A Case Study on PPO and TRPO:
Overall, our results on the importance of these optimizations both corroborate results demonstrating the brittleness of deep policy gradient methods, and demonstrate that even beyond environmental brittleness, the algorithms themselves exhibit high sensitivity to implementation choices

[FORTECH.AI] Links – DRL – Applications

Deep Reinforcement Learning: From Toys to Enteprise
Applications of Reinforcement Learning in Real World

Learning Robust Control Policies for End-to-End Autonomous Driving From Data-Driven Simulation
"To the best of our knowledge, this work is the first published report of a full-scale autonomous vehicle trained entirely in simulation using only reinforcement learning, that is capable of being deployed onto real roads and recovering from complex, near crash driving scenarios."

[FORTECH.AI] References

1. A3C: Asynchronous methods for deep reinforcement learning.
2. DQN: Human-level control through deep reinforcement learning.
3. IMPALA: IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures.
4. APE-X: Distributed prioritized experience replay.
5. Accel RL: Accelerated methods for deep reinforcement learning.
6. PBT: Population based training of neural networks.
7. AlphaGo: Mastering the game of Go without human knowledge.
8. AlphaGo Zero Blog: https://deepmind.com/blog/alphago-zero-learning-scratch/
9. CTF: Human-level performance in first-person multiplayer games with population-based deep reinforcement learning.
10. Dota2 Blog: https://blog.openai.com/openai-five/
11. PPO: Proximal policy optimization algorithms.

[FORTECH.AI] Links – DRL – Other Papers

Deep Recurrent Q-Learning for Partially Observable MDPs
Deep Q-learning from Demonstrations

[FORTECH.AI] Links – DRL – Challenges

NeurIPS RL Competitions Results Presentations: MineRL (MineRL: Towards AI in Minecraft)
Hierarchical Deep Q-Network from Imperfect Demonstrations in Minecraft
The MineRL competition
AWS JPLOPEN SOURCE ROVER CHALLENGE

[FORTECH.AI] Links – Startups - DRL in Robotics

Bonsai (Industrial Robotic Arms with Hierarchical RL):
Microsoft's AI Platform Gets A Big Boost With Bonsai Acquisition
Concept Network Reinforcement Learning for Flexible Dexterous Manipulation

Covariant (Industrial Robotic Arms):
AI-powered robot warehouse pickers are now ready to go to work
Covariant Uses Simple Robot and Gigantic Neural Net to Automate Warehouse Picking

Skydio (consumer drone):
Deep Neural Pilot on Skydio 2
Skydio crams all its autonomous magic into a sleek consumer drone that costs under $1k

[FORTECH.AI] RL Overview – Cheat Sheets

http://www.montreal.ai/ai4all.pdf
A Map of Reinforcement Learning
Reinforcement Learning Cheat Sheet
CS 287 Advanced Robotics – Fundamental Knowledge

Thank You!

We ❤ feedback!