Multi-task DeepRL

- General agents that are able to do many tasks simultaneously.
- Going from one network per task to one network for tens of tasks with many challenges.
- Data Efficiency: Hundreds of millions of frames for a single task.
- Stability: Do we need task-specific hyperparameters?
- Scale: More complicated architecture and slower to train.
- Task Interference: Will multiple tasks cause interference or positive transfer.
Agent interacting with the environment. At each step $t$:

1. Agent takes action $a_t$
2. Environment returns reward $r_{t+1}$ and state $s_{t+1}$

Maximize total future reward

$$r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots$$

For a policy $\pi$ the action value function $Q$:

$$Q^\pi(s, a) = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s, a_t = a]$$

$$= \mathbb{E}[r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) | s_t = s, a_t = a]$$

$Q$ represents how good an action $a$ is given state $s$. 
Optimal Value Functions

- An optimal value function gives the maximal achievable value:

\[ Q^*(s, a) = \max_\pi Q^\pi(s, a) \]

- Given an optimal value function we can get an optimal policy:

\[ \pi^*(s) = \arg \max_a Q^*(s, a) \]

- Optimal value functions also obey a Bellman Equation.

\[ Q^*(s_t, a_t) = \mathbb{E}[r_t + \gamma \max_{s'} Q^*(s_{t+1}, a')] \]
• High-level idea is to make Q-learning look like supervised learning
• Optimize the Q-learning loss with minibatch SGD
• Apply Q-learning updates on batches of past experience instead of online
  1. Experience replay
  2. Previously used for better data efficiency
  3. Makes the data distribution more stationary
• Use an older set of weights to compute the targets, keeps the target function from changing too quickly

\[ L_i(\theta_i) = \mathbb{E}_{s,a,s',r \sim D} (r + \gamma \max a' Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2 \]
Policy Gradient Methods

- An alternatively class of methods directly optimize the expected return of a policy:

\[ \nabla_\theta J(\theta) = \nabla E[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots] \]

- For all differentiable policies

\[ \nabla_\theta J(\theta) = E[\nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s, a)] \]

where expectations is over states and actions.

- There is an sample based easy unbiased estimation (REINFORCE)

\[ \nabla_\theta \log \pi_\theta(a \mid s) R_t \]

where

\[ R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \]
• Start with a guess for each \( Q(s, a) \)

• Interact with the environment using some policy based on \( Q \) collecting tuples of experience \( \{s_t, a_t, r_t, s_{t+1}, \cdots\} \), e.g. \( \varepsilon \)-greedy.

• Apply updates based on the Bellman equation

\[
Q(s, a) \leftarrow Q(s, a) + (r + \gamma \max_a Q(s', a') - Q(s, a))
\]

• \( Q(s, a) \) is guaranteed to converge to the optimal value function \( Q^* \) under some reasonable assumptions.
Asynchronous Advantage Actor-Critic (A3C)

- The agent learns a policy and a state value function
- Uses bootstrapped n-step returns to reduce variance over REINFORCE with a baseline
- The policy gradient multiplied by an estimate of the advantage. Similar to Generalized Advantage Estimation (Schulman et al. 2015)

\[
\nabla_\theta \log \pi (a_t \mid s_t, \theta) \left( \sum_{k=0}^{N} \gamma^k r_{t+k} + \gamma^{N+1} V(s_{t+N+1}) - V(s_t) \right)
\]

- The critic/value function is trained with n-step TD learning. i.e. by minimizing the MSE

\[
\left( \sum_{k=0}^{N} \gamma^k r_{t+k} + \gamma^{N+1} V(s_{t+N+1}; \theta^-) - V(s_t; \theta) \right)^2
\]
Our goal was to scale up A3C since it has more of the desired properties of a good multi-task agent.

- Adding more actor/learners does not scale.
- Distributed experience collection is good.
- Communicating gradients is bad.
A Better Architecture

- It is better to use a centralized learner(s) and distribute the acting
- Actors receive parameters but send observations
- The centralized learner can parallelize as much of the forward and backward passes as possible
Figure 2. Timeline for one unroll with 4 steps using different architectures. Strategies shown in (a) and (b) can lead to low GPU utilisation due to rendering time variance within a batch. In (a), the actors are synchronised after every step. In (b) after every $n$ steps. IMPALA (c) decouples acting from learning.
Decoupled Backward Pass

- It is more efficient to decouple the backward pass
- Actors generate trajectories/unrolls and place them into a queue
- The learner continuously dequeues batches of trajectories and performs parameter updates

Key Challenge:
1. Decoupling the backwards pass requires off-policy learning
2. Actor parameters can lag by several updates

POD architecture-Parallel Off-policy Decoupled
The experience generated by the actors can lag behind the learner’s policy.

We introduce a principled off-policy advantage actor critic called V-Trace.

The V-Trace corrected estimate for the value $V(x_s)$ is:

$$v_s \overset{\text{def}}{=} V(x_s) + \sum_{t=s}^{s+n-1} \gamma^{t-s} \left( \prod_{i=s}^{t-1} c_i \right) \rho_t (r_t + \gamma V(x_{t+1}) - V(x_t)),$$

where $\rho_t \overset{\text{def}}{=} \min (\bar{\rho}, \pi(a_t|x_t)/\mu(a_t|x_t))$ and $c_i \overset{\text{def}}{=} \min (\bar{c}, \pi(a_i|x_i)/\mu(a_i|x_i))$.

The V-Trace update for the value function is:

$$(v_s - V_\theta(x_s)) \nabla_\theta V_\theta(x_s)$$

The V-Trace update for the policy is:

$$\rho_s \nabla_\omega \log \pi_\omega(a_s|x_s) \left( r_s + \gamma v_{s+1} - V_\theta(x_s) \right)$$
Now in the off-policy setting that we consider, we can use an IS weight between the policy being evaluated $\pi_{\bar{\rho}}$ and the behaviour policy $\mu$, to update our policy parameter in the direction of

$$\mathbb{E}_{a_s \sim \mu(\cdot|x_s)} \left[ \frac{\pi_{\bar{\rho}}(a_s|x_s)}{\mu(a_s|x_s)} \nabla \log \pi_{\bar{\rho}}(a_s|x_s) q_s|x_s \right]$$

where $q_s \overset{\text{def}}{=} r_s + \gamma v_{s+1}$ is an estimate of $Q^{\pi_{\bar{\rho}}}(x_s,a_s)$ built from the V-trace estimate $v_{s+1}$ at the next state $x_{s+1}$. The reason why we use $q_s$ instead of $v_s$ as the target for our Q-value $Q^{\pi_{\bar{\rho}}}(x_s,a_s)$ is that, assuming our value estimate is correct at all states, i.e. $V = V^{\pi_{\bar{\rho}}}$, then we have $\mathbb{E}[q_s|x_s,a_s] = Q^{\pi_{\bar{\rho}}}(x_s,a_s)$ (whereas we do not have this property if we choose $q_t = v_t$).
Define the V-trace operator $\mathcal{R}$:

$$
\mathcal{R}V(x) \overset{\text{def}}{=} V(x) + \mathbb{E}_\mu \left[ \sum_{t \geq 0} \gamma^t (c_0 \ldots c_{t-1}) \rho_t (r_t + \gamma V(x_{t+1}) - V(x_t)) \right],
$$

**Theorem**

Let $\rho_t = \min (\bar{\rho}, \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)})$ and $c_t = \min (\bar{c}, \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)})$ be truncated importance sampling weights, with $\bar{\rho} \geq \bar{c}$. Assume that there exists $\beta \in (0, 1]$ such that $\mathbb{E}_\mu \rho_0 \geq \beta$. Then the operator $\mathcal{R}$ defined by (3) has a unique fixed point $V^{\pi \bar{\rho}}$, which is the value function of the policy $\pi \bar{\rho}$ defined by

$$
\pi \bar{\rho}(a|x) \overset{\text{def}}{=} \frac{\min (\bar{\rho} \mu(a|x), \pi(a|x))}{\sum_{b \in A} \min (\bar{\rho} \mu(b|x), \pi(b|x))},
$$

(4)

Furthermore, $\mathcal{R}$ is a $\eta$-contraction mapping in sup-norm, with

$$
\eta \overset{\text{def}}{=} \gamma^{-1} - (\gamma^{-1} - 1) \mathbb{E}_\mu \left[ \sum_{t \geq 0} \gamma^t \left( \prod_{i=0}^{t-2} c_i \right) \rho_{t-1} \right] \leq 1 - (1 - \gamma) \beta < 1.
$$
Theorem

Assume a tabular representation, i.e. the state and action spaces are finite. Consider a set of trajectories, with the \( k^{th} \) trajectory \( x_0, a_0, r_0, x_1, a_1, r_1, \ldots \) generated by following \( \mu: a_t \sim \mu(\cdot|x_t) \). For each state \( x_s \) along this trajectory, update

\[
V_{k+1}(x_s) = V_k(x_s) + \alpha_k(x_s) \sum_{t \geq s} \gamma^{t-s} (c_s \ldots c_{t-1}) \rho_t (r_t + \gamma V_k(x_{t+1}) - V_k(x_t))
\]

(5)

with \( c_i = \min(\overline{c}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)}) \), \( \rho_i = \min(\overline{\rho}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)}) \), \( \overline{\rho} \geq \overline{c} \). Assume that (1) all states are visited infinitely often, and (2) the stepsizes obey the usual Robbins-Munro conditions: for each state \( x \), \( \sum_k \alpha_k(x) = \infty \), \( \sum_k \alpha_k^2(x) < \infty \). Then \( V_k \to V^{\pi\overline{\rho}} \) almost surely.
Figure 1: Model Architectures. Left: Small architecture, 2 convolutional layers and 1.2 million parameters. Right: Large architecture, 15 convolutional layers.
## Throughput Comparison

<table>
<thead>
<tr>
<th>Architecture</th>
<th>CPUs</th>
<th>GPUs$^1$</th>
<th>FPS$^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single-Machine</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3C 32 workers</td>
<td>64</td>
<td>0</td>
<td>6.5K</td>
</tr>
<tr>
<td>Batched A2C (sync step)</td>
<td>48</td>
<td>0</td>
<td>9K</td>
</tr>
<tr>
<td>Batched A2C (sync step)</td>
<td>48</td>
<td>1</td>
<td>13K</td>
</tr>
<tr>
<td>Batched A2C (sync traj.)</td>
<td>48</td>
<td>0</td>
<td>16K</td>
</tr>
<tr>
<td>Batched A2C (dyn. batch)</td>
<td>48</td>
<td>1</td>
<td>16K</td>
</tr>
<tr>
<td>IMPALA 48 actors</td>
<td>48</td>
<td>0</td>
<td>17K</td>
</tr>
<tr>
<td>IMPALA (dyn. batch) 48 actors$^3$</td>
<td>48</td>
<td>1</td>
<td>21K</td>
</tr>
<tr>
<td><strong>Distributed</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3C</td>
<td>200</td>
<td>0</td>
<td>46K</td>
</tr>
<tr>
<td>IMPALA</td>
<td>150</td>
<td>1</td>
<td>80K</td>
</tr>
<tr>
<td>IMPALA (optimised)</td>
<td>375</td>
<td>1</td>
<td>200K</td>
</tr>
<tr>
<td>IMPALA (optimised) batch 128</td>
<td>500</td>
<td>1</td>
<td>250K</td>
</tr>
</tbody>
</table>

1. Nvidia P100  
2. In frames/sec (4 times the agent steps due to action repeat).  
3. Limited by amount of rendering possible on a single machine.
**Figure 4. Top Row:** Single task training on 5 DeepMind Lab tasks. Each curve is the mean of the best 3 runs based on final return. IMPALA achieves better performance than A3C. **Bottom Row:** Stability across hyperparameter combinations sorted by the final performance across different hyperparameter combinations. IMPALA is consistently more stable than A3C.
DMLab-30 Multi-Task Results

![Graph showing performance of different algorithms over environment frames.](image)

- **Mean Capped Normalized Score**
- **Environment Frames**
- Algorithms: IMPALA, deep, PBT - 8 GPUs, IMPALA, shallow, IMPALA, deep, IMPALA-Experts, deep, A3C, deep
Conclusions

- New distributed RL architecture allows for stable learning with very high throughput
- Especially well-suited to the multi-task deep RL setting
- Synchronous batch learning is more robust to hyperparameters than async SGD
- Multi-task RL on the DMLab-30:
  1. Positive transfer
  2. Deep ResNets finally outperforms 3 layer ConvNets (Atari was too simple)