Action Branching Architectures for Deep Reinforcement Learning

Arash Tavakoli   Fabio Pardo   Petar Kormushev

Imperial College London

The Thirty-Second AAAI Conference on Artificial Intelligence
New Orleans, Louisiana, USA | February 2018

Motivation

Discrete-action algorithms have been central to numerous recent successes of deep reinforcement learning

Deep Q-Networks (DQN):

- State-of-the-art performance on the Atari 2600 benchmark
- Many extensions (e.g. Distributional and Double Q-Learning)
- Off-policy algorithm

Major Disadvantage:

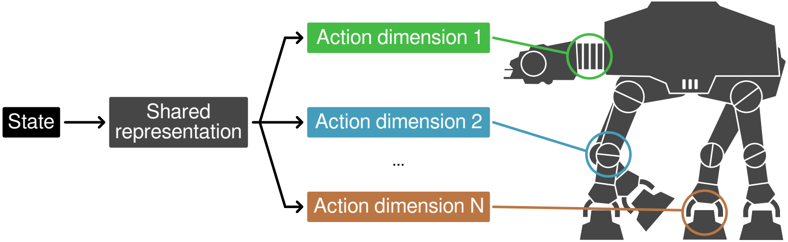
- Direct application to domains with high-dimensional discrete or continuous action spaces is considered intractable

Problem

Combinatorial increase of the number of actions
with increasing action dimensionality

- Such large action spaces are difficult to explore efficiently
- Training is computationally expensive

Action Branching Architecture



Achieves linear increase vs. combinatorial increase of the number of outputs

- Each action branch controls an individual degree of freedom
- Optimize for each action dimension with some independence

The Shared Network Module

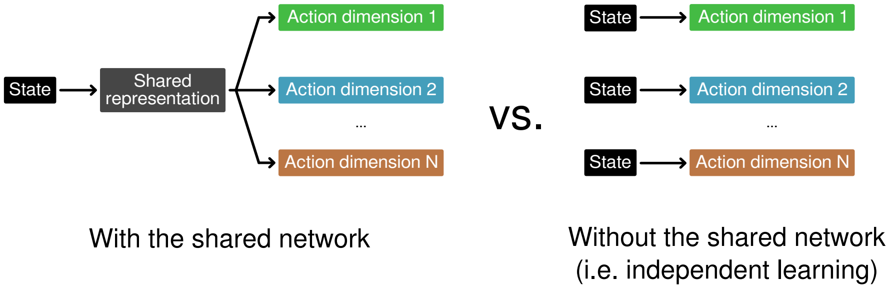
Without the shared network module,
learning is subject to coordination issues

Hypothesis

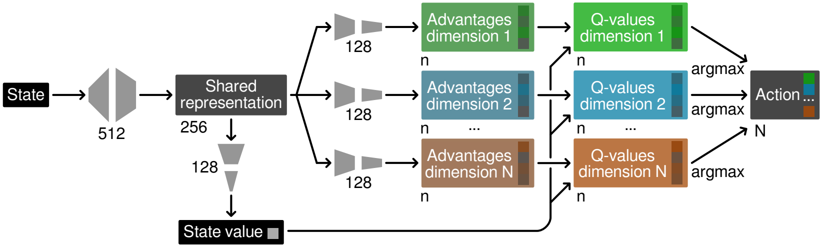
Due to the backpropagation of the gradients originating from all branches, the shared network module can help coordinate the action branches and stabilize training.

Empirical Verification

We verify the hypothesis via an ablation study by comparing the performance of an action branching agent *with* and *without* the shared network module



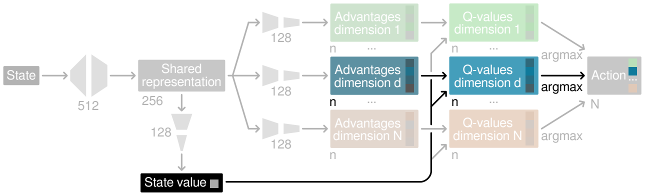
We design a proof-of-concept, action branching agent



- Select DQN as the algorithmic basis
- Adapt Double Q-Learning, Dueling Architectures, and Prioritized Replay

Branching Dueling Q-Network

→ Common State-Value Estimation



Aggregation Methods:

- Naïve aggregation: $Q_d(s, u_d) = V(s) + A_d(s, u_d)$
- Locally subtract each branch's mean advantage (*best-performing*):

$$Q_d(s, u_d) = V(s) + \left(A_d(s, u_d) - \frac{1}{n} \sum_{u'_d \in \mathcal{U}_d} A_d(s, u'_d)\right)$$

Branching Dueling Q-Network

→ Temporal-Difference Target

- Double DQN targets for each branch **separately**:

$$y_d = r + \gamma Q_d^-(s', \arg\max_{u'_d \in \mathcal{U}_d} Q_d(s', u'_d))$$

- Mean Double DQN target across all action branches as a single **global target** for all action branches (*best-performing*):

$$y = r + \gamma \frac{1}{N} \sum_d Q_d^-(s', \arg\max_{u'_d \in \mathcal{U}_d} Q_d(s', u'_d))$$

Branching Dueling Q-Network

→ Aggregation of Temporal-Difference Errors

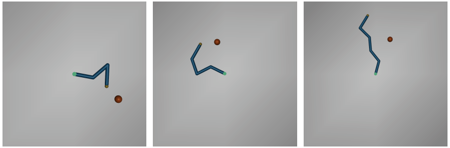
Loss Function:

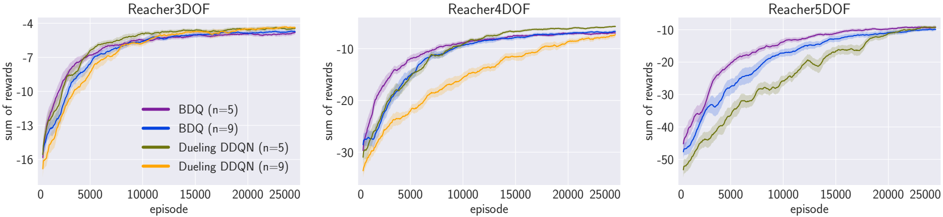
$$L = \mathbb{E}_{(s,u,r,s')\sim\mathcal{D}}\left[\frac{1}{N}\sum_{d}\left(y_d - Q_d(s, u_d)\right)^2\right]$$

Error for Experience Prioritization:

$$e_D(s, u, r, s') = \sum_{d} \left| y_d - Q_d(s, u_d) \right|$$

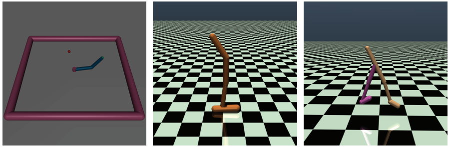
To *branch*, or not to *branch*: that is the question

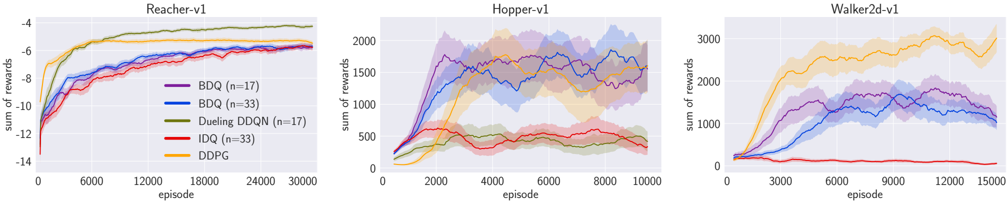




- BDQ (branching) robustly scales with increasing action dimensionality and discretization granularity, while Dueling DDQN (non-branching) quickly deteriorates
- Dueling DDQN becomes computationally expensive for 5 joints and 9 sub-actions per joint (i.e. 59049 actions at output)

With vs. *Without* the Shared Network Module

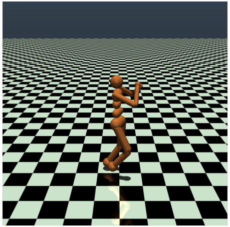


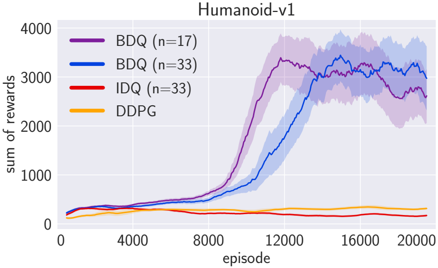


- BDQ (with shared network) robustly scales with increasing action dimensionality, while its independent variant IDQ (without shared network) quickly deteriorates
- BDQ performs robustly with varying discretization granularity

Continuous-Action vs. Branching *Discrete-Action* Methods

for Continuous Control





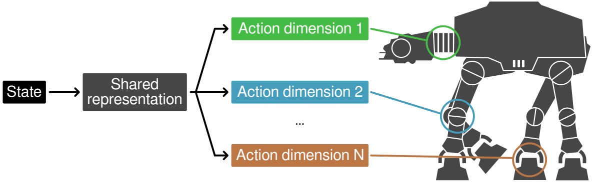
- BDQ efficiently learns to solve the Humanoid-v1 domain with a total of 6.5×10^{25} possible discrete actions
- BDQ learns robustly w.r.t. the discretization granularity (17 and 33 sub-actions per action dimension)

Summary and Conclusion

1. Introduced a neural architecture (**action branching**) for enabling the application of discrete-action algorithms to high-dimensional domains
2. Hypothesized the significance of the shared network in coordinating the branches
3. Described a novel agent (**BDQ**) based on the incorporation of the action branching architecture into the DQN algorithm
4. Empirically verified the hypothesis and illustrated the effectiveness of action branching in tackling domains with as many as 6.5×10^{25} possible actions
5. Potentially applicable to other discrete-action algorithms

Beyond this talk...

Action Branching Architectures
for Deep Reinforcement Learning



- Code: `github.com/atavakol/action-branching-agents`
- Email: `a.tavakoli@imperial.ac.uk`
- Twitter: `@arshtvk`