⚔️

Dueling DQN

Paper Link: Dueling Network Architectures for Deep Reinforcement Learning

Key Features

- The Dueling Network Architecture proposes a network architecture better suited for model-free RL than the existing standard neural network.
- The Dueling network architecture estimates the state-action value function Q using two streams consisting the state value and action advantage functions while having one common CNN network.
- The Dueling architecture can be combined with various existing and future model free RL algorithms.

Background

This paper proposes a novel neural network architecture suitable for reinforcement learning instead of the standard neural network architecture that uses a single stream of neural networks. The Dueling network architecture as shown in Figure1 consists of two streams that represent the state value and action advantage functions, while sharing a common convolutional feature learning module.

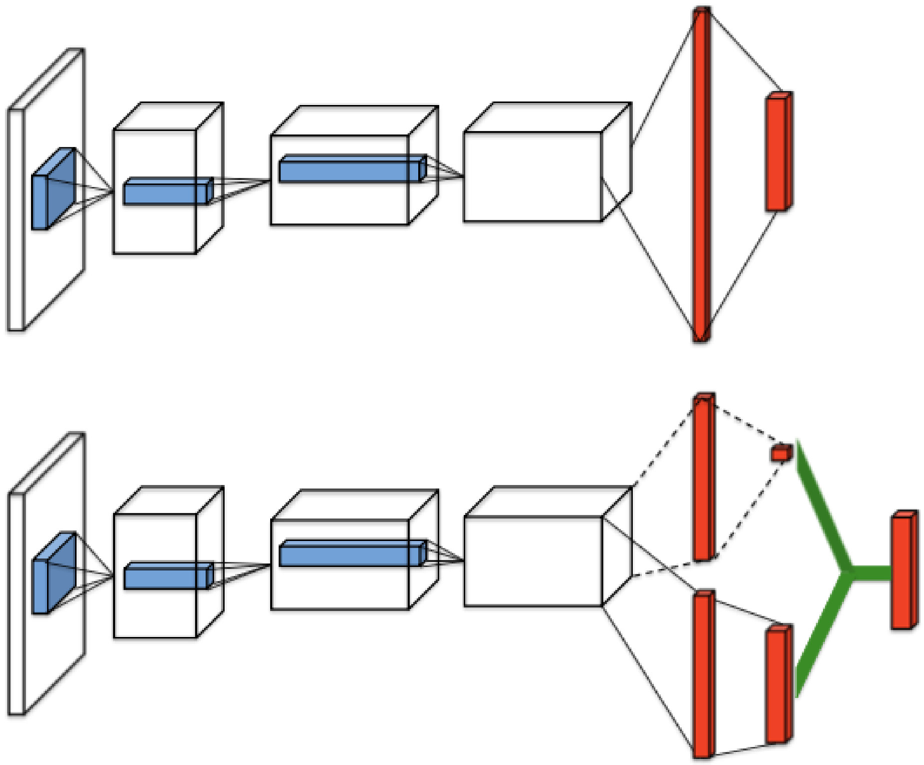

Figure 1. A popular single stream Q-network (**top**) and the dueling Q-network (**bottom**). The dueling network has two streams to separately estimate (scalar) state-value and the advantages for each action; the green output module implements equation (9) to combine them. Both networks output Q-values for each action.

For an agent behaving according to a stochastic policy π, the values of the state-action pair (s, a) and the state s are defined as follows:

$$Q^{\pi}(s,a) = \mathbb{E}[R_t | s_t = s, a_t = a, \pi],$$
$$V^{\pi}(s) = \mathbb{E}_{a \sim \pi(s)}[Q^{\pi}(s,a)]$$

The paper defines another important quantity, the advantage function, relating the value and Q functions: $A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$.

Note that $E_{a \sim \pi(s)}[A^{\pi}(s,a)] = 0$. Intuitively, the state value function V measures the value of a particular state s. The Q function, however, measures the the value of the particular state and action. The advantage function subtracts the value of the state from the Q function to obtain a relative measure of the importance of each action.

Method

The Dueling Network Architecture

From the expressions for advantage $A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$ and state-value $V^{\pi}(s) = \mathbb{E}_{a\sim\pi(s)}[Q^{\pi}(s,a)]$, it follows that $\mathbb{E}_{a\sim\pi(s)}[A^{\pi}(s,a)] = 0$.

Moreover, for a deterministic policy, $a^* = \arg\max_{a'\in A} Q(s,a')$, it follows that $Q(s,a^*) = V(s)$ and hence $A(s,a^*) = 0$.

According to the above, the network in Figure1 can be expressed as an equation as follows.

$$Q(s,a;\theta,\alpha,\beta) = V(s,\theta,\beta) + A(s,a;\theta,\alpha)$$

However, the above equation has two problems. First, the $Q(s,a;\theta,\alpha,\beta)$ is only a parameterized an estimate of the true Q-function. Second, given Q is unidentifiable in that it cannot uniquely recover V and A.

To solve the above problem, when the optimal action $a^* = \arg\max_{a\prime\in A} Q(s,a\prime;\theta,\alpha,\beta)$ is chosen, $Q(s,a^*;\theta,\alpha,\beta)$ is made equal to $V(s;\theta,\beta)$. This allows the advantage function estimator can force the advantage to have zero advantage at the chosen action. This method can be Equation as

$$Q(s,a;\theta,\alpha,\beta) = V(s;\alpha,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a'\in|A|} A(s,a';\theta,\alpha)\right)$$

With this, the stream $V(s;\theta,\beta)$ provides an estimate of the value function, while the other stream produces an estimate of the advantage function.

An alternative module replaces the max operator instead of an average:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\alpha,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)\right)$$

The above equation loses its original meaning because V and A are now off-target by constants, but on the other hand it increases the stability of the optimization.

Algorithm

Dueling DQN algorithm uses the algorithm of Double DQN as it is, and only the network is replaced by the Dueling network architecture.

Algorithm 1: Double DQN Algorithm.

input : $\mathcal{D}$ – empty replay buffer; θ – initial network parameters, θ^- – copy of θ
input : N_r – replay buffer maximum size; N_b – training batch size; N^- – target network replacement freq.
for *episode* $e \in \{1, 2, \ldots, M\}$ **do**
  Initialize frame sequence $\mathbf{x} \leftarrow ()$
  for $t \in \{0, 1, \ldots\}$ **do**
    Set state $s \leftarrow \mathbf{x}$, sample action $a \sim \pi_{\mathcal{B}}$
    Sample next frame x^t from environment $\mathcal{E}$ given (s, a) and receive reward r, and append x^t to $\mathbf{x}$
    if $|\mathbf{x}| > N_f$ **then** delete oldest frame $x_{t_{min}}$ from $\mathbf{x}$ **end**
    Set $s' \leftarrow \mathbf{x}$, and add transition tuple (s, a, r, s') to $\mathcal{D}$,
      replacing the oldest tuple if $|\mathcal{D}| \geq N_r$
    Sample a minibatch of N_b tuples $(s, a, r, s') \sim \text{Unif}(\mathcal{D})$
    Construct target values, one for each of the N_b tuples:
    Define $a^{\max}(s'; \theta) = \arg\max_{a'} Q(s', a'; \theta)$
    $$y_j = \begin{cases} r & \text{if } s' \text{ is terminal} \\ r + \gamma Q(s', a^{\max}(s'; \theta); \theta^-), & \text{otherwise.} \end{cases}$$
    Do a gradient descent step with loss $\| y_j - Q(s, a; \theta) \|^2$
    Replace target parameters $\theta^- \leftarrow \theta$ every N^- steps
  end
end

Implementation on JORLDY

- In JORLDY, only the network is replaced with the DQN algorithm.
- Dueling DQN Agent Implementation

```
from .dqn import DQN

class Dueling(DQN):
    def __init__(self, *args, **kwargs):
        if "network" in kwargs.keys():
            kwargs["network"] = "dueling"
        assert kwargs["network"] == "dueling"
        super(Dueling, self).__init__(*args, **kwargs)
```

- Dueling DQN Network Implementation

```
class Dueling(BaseNetwork):
    def __init__(self, D_in, D_out, D_hidden=512, head="mlp"):
        D_head_out = super(Dueling, self).__init__(D_in, D_hidden, head)

        self.l1_a = torch.nn.Linear(D_head_out, D_hidden)
        self.l1_v = torch.nn.Linear(D_head_out, D_hidden)

        self.l2_a = torch.nn.Linear(D_hidden, D_out)
        self.l2_v = torch.nn.Linear(D_hidden, 1)

        orthogonal_init([self.l1_a, self.l1_v])
        orthogonal_init([self.l2_a, self.l2_v], "linear")

    def forward(self, x):
        x = super(Dueling, self).forward(x)

        x_a = F.relu(self.l1_a(x))
        x_v = F.relu(self.l1_v(x))

        # A stream : action advantage
        x_a = self.l2_a(x_a)  # [bs, num_action]
        x_a -= x_a.mean(dim=1, keepdim=True)  # [bs, num_action]

        # V stream : state value
        x_v = self.l2_v(x_v)  # [bs, 1]

        out = x_a + x_v  # [bs, num_action]
        return out
```

References

Relevant papers

- Multi-Player Residual Advantage Learning With General Function Approximation
- Deep Reinforcement Learning with Double Q-learning