Value-based Deep Reinforcement Learning

基于值的深度强化学习。

这一部分我们介绍三种常见的基于值的深度强化学习(RL)算法:深度 Q-网络(Deep Q-network)[1]，双 DQN(Double DQN)[2]和对抗 DQN(Dueling DQN)[3]。通过端到端的(end-to-end)强化学习，这三种神经网络架构都可以直接从像电子游戏的预处理像素这样的高维输入(high-dimensional inputs)中学习到成功的策略，它们在 Atari 2600 的 49 个游戏中达到了相当于专业的游戏测试员的水平[4]。

这些架构都使用了卷积神经网络(Convolutional Neural Networks, CNNs)[5]来从像素输入中提取特征。理解 CNNs 特征提取的机制有助于我们理解 DQN 的工作方式。

Recap:Action-Value Function Approximation

状态-行为值函数近似。

上节课我们用参数化的近似函数来表示状态-行为值函数(Q-函数)，若我们将参数记为$\mathbf{W}$，那么这种设定中，Q-函数可以被表示为$\hat{q}(s,a,\mathbf{w})$。

假设我们知道$q(s,a)$，通过最小化真实状态-动作值函数$q(s,a)$和近似估计之间的均方误差$J(\mathbf{w})$，我们可以得到近似的 Q-函数:

$$J(\mathbf{w}) = \mathbb{E}[(q(s,a) - \hat{q}(s,a,\mathbf{w}))^2] \tag{1}$$

我们可以通过随机梯度下降(stochastic gradient descent, SGD)找到 J 的关于 w 的局部最小值，并且以下式来更新 w:

$$\Delta(\mathbf{w}) = -\frac{1}{2}\alpha\nabla_{\mathbf{w}}J(\mathbf{w}) = \alpha\mathbb{E}[(q(s,a) - \hat{q}(s,a,\mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(s,a,\mathbf{w})] \tag{2}$$

α为学习率(learning rate)。通常情况下,真实的状态-动作值函数 q(s,a)是未知的,所以用一个近似的学习目标(learning target)来代替公式(2)中的 q(s,a)。

在蒙特卡洛方法中,对于片段式的 MDPs,我们用无偏的回报G_t来代替学习目标:

$$\Delta(\mathbf{w}) = \alpha(G_t - \hat{q}(s,a,\mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(s,a,\mathbf{w}) \tag{3}$$

对于 SARSA 方法，我们使用基于推导的 TD 目标$r + \gamma\hat{q}(s',a',\mathbf{w})$:

$$\Delta(\mathbf{w}) = \alpha(r + \gamma\hat{q}(s',a',\mathbf{w}) - \hat{q}(s,a,\mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(s,a,\mathbf{w}) \tag{4}$$

这里 a'为在下一状态 s'执行的动作,γ为衰减因子,TD 目标利用了当前的函数近似值。

对于 Q 学习，我们使用 TD 目标$r + \gamma\max_{a'}\hat{q}(s',a',\mathbf{w})$,并以下式来更新 w:

$$\Delta(\mathbf{w}) = \alpha(r + \gamma\max_{a'}\hat{q}(s',a',\mathbf{w}) - \hat{q}(s,a,\mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(s,a,\mathbf{w}) \tag{5}$$

下面的部分我们将介绍如何使用深度神经网络来近似$\hat{q}(s,a,\mathbf{w})$，以及如何通过端到端的训练来学习神经网络的参数 w。

Generalization:Deep Q-network

泛化深度 Q-网络。

线性近似函数的表现非常依赖于特征的质量，一般来说，手动给出合适的特征很困难，也很耗时。为了在大的空间(large domians)(比如大的状态空间)进行决策并实现特征自动提取，深度神经网络 (DNNs)通常被选作近似函数。

DQN Architecture

图 1 展示了 DQN 的结构，图中的网络将 Atari 游戏环境的预处理像素图像(预处理见 7.2.2 节)作为输入,为每个可行的动作赋予一个 Q 值,将这些 Q 值作为一个向量输出。预处理的像素输入代表了游戏状态 s,单个的输出单元表示动作 a 的$\hat{q}$函数。总的来说，$\hat{q}$可以被记为 $\hat{q}(s, \mathbf{w}) \in \mathbb{R}^{|A|}$，简单起见，我们仍用$\hat{q}(s, a, \mathbf{w})$来表示对(s,a)的状态-行为值估计。

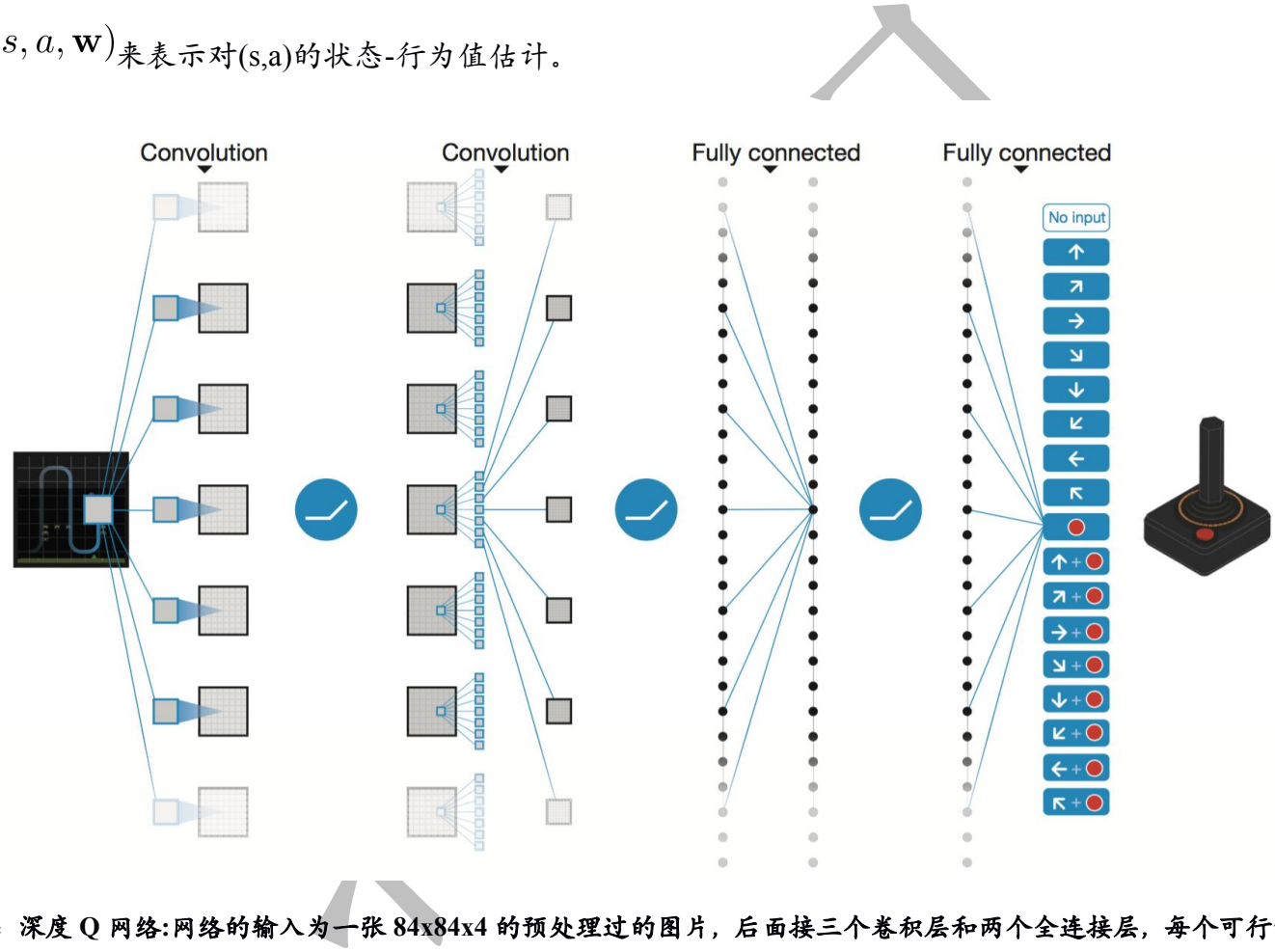

图 1：深度 Q 网络:网络的输入为一张 84x84x4 的预处理过的图片，后面接三个卷积层和两个全连接层，每个可行动作都对应一个输出，每个隐藏层都接一个非线性整流器(ReLU)。

结构细节：输入为一张 84x84x4 的图片;第一个卷积层有 32 个大小为 8x8、步幅（stride）为 4 的滤波器，对输入图片进行卷积操作后，将结果输入非线性整流器(ReLU);第二个隐藏层有 64 个大小为 4x4、步幅为 2 的滤波器，同样地，后面接非线性整流器；第三个隐藏层有 64 个大小为 3x3、步幅为 1 的滤波器，同样地，后面接 ReLU；最后的隐藏层为包含 512 个整流器(ReLU)单元的全连接层(fully-connected layer);输出层为全连接线性层。

Preprocessing Raw Pixels

预处理原始图像,原始 Atari 2600 帧的尺寸为(210 x 160 x 3),最后一个维度对应于 RGB 通道。[1]中采

用的预处理步骤旨在减小输入维数并处理游戏模拟器的一些工件。我们将预处理总结如下:

- 单帧编码(single frame encoding):要编码单个帧,则在要编码的帧上返回每个像素颜色值的最大值,并返回前一帧。换句话说,我们返回 2 个连续的原始像素帧的逐像素最大池化(pixel-wise max-pooling)。
- 降维:从编码的 RGB 帧中提取 Y 通道,也称为亮度,并将其重新缩放为(84×84×1)。

将上述预处理应用于最近的 4 个原始 RGB 帧,并将编码后的帧堆叠在一起以生成 Q-Network 的输入(形状(84x84x4))。将最近的帧堆叠在一起作为游戏状态也是将游戏环境转变成(几乎)马尔可夫世界的一种方式(当前图像仅和上一幅图像有关)。

Training Algorithm for DQN

由于理论上的保证是不可能的,并且学习和训练往往非常不稳定,过去常常避免使用大型深层神经网络函数逼近器来学习动作值函数。为了使用大型非线性函数逼近器和扩展在线 Q 学习,DQN 引入了两个主要更改:使用经验回播(experience replay)和单独的目标网络(target network)。完整的算法在算法 1 中介绍。本质上,通过最小化以下均方误差来学习 Q 网络:

$$J(w) = \mathbb{E}_{s_t,a_t,r_t,s_{t+1}}\left[y_t^{DQN} - \hat{q}(s_t,a_t,w))^2\right] \qquad (6)$$

其中y_t^{DQN}是向前一步的学习目标:

$$y_t^{DQN} = r_t + \gamma \max_{a'} \hat{q}(s_{t+1},a',w-) \qquad (7)$$

其中 w−表示 target newwork 的参数,online network 的参数 w 通过对过去的过渡元组(transition tuples< s_t,a_t,r_t,s_{t+1}>采样后进行小批量梯度更新(这里的意思应该是进行自举,也就是先收集一大堆刚才提到的元组数据,然后再对这些元组进行小批量的采样作为数据)。(注意:尽管学习目标是使用参数为 w-的 target network 计算出来的,但目标y_t^{DQN}在对 w 进行更新时被认为是固定的)

Experience replay:

下列的时间步(time step)是指:自定义的单位时间,比如 1 秒收集一次,或者代码运行一次收集一次,并且上诉的经验,experience 可以理解为历史数据。

agent 在每个时间步的经验(或称转换,transition)$e_t = (s_t,a_t,r_t,s_{t+1})$都被存储在固定大小的数据集(或重播缓冲区,replay buffer)$D_t = \{e_1,\ldots e_t\}$。重播缓冲区用于存储最近的 k = 1 百万次经验(重播缓冲区的说明请参见图 2)。Q 网络的参数更新是通过对小批量数据采样后进行 SGD 更新。

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
...
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow \quad s, a, r, s'$

在小批量数据中的每个历史数据(transition sample)都是从存储的经验库(s,a,r,s')~U(D)中随机地均匀采样。与标准的在线 Q 学习相比,此方法具有以下优点:

- 更高的数据效率：历史数据的每个步骤都可以用于许多更新，从而提高了数据效率。
- 消除样本相关性：随机抽取的历史数据(randomizing the transition experience)会破坏连续样本之间的相关性，因此会减少更新的方差并稳定学习。
- 避免振荡或发散：行为分布是在其许多先前状态和转换中求平均的，从而使学习变得平滑，并避免了参数的振荡或发散。(请注意，使用经验回放时，必须使用 off-policy 方法，例如 Q 学习，因为当前参数与用于生成样本的参数不同)

Target network:

为了进一步提高学习的稳定性并处理非平稳的学习目标，在 Q 学习更新中，使用了一个单独的目标网络来生成目标y_j(算法 1 中的第 12 行)。更具体地，每次 C 更新 t 都是通过从 online network q^(s, a, w) 复制参数值($w^- =$ w)来更新 target network $q\hat{}(s, a, w^-)$,target network 保持不变，并随 C 更新生成目标y_j。与标准的在线 Q 学习相比，此修改使算法更稳定，原始 DQN 中的 C = 10000。

Training Details

在原始 DQN 论文[1]中，在具有相同架构，学习算法和超参数的每个游戏上训练了不同的网络（或 agent)。作者将来自游戏环境的所有正面奖励削减为+1，将所有负面奖励削减为-1，这使得在所有不同游戏中使用相同的学习率成为可能。对于具有生命计数器（例如 Breakout）的游戏，仿真器还会返回游戏中剩余的生命数，然后将其用于在训练期间通过将未来奖励明确设置为零来标记情节的结束。他们还使用了一种简单的跳帧技术（或动作重复）：代理在第 4 帧而不是每帧上选择动作，并且在跳过的帧上重复其最后一个动作。这减少了决策的频率不会对性能造成太大影响，并且使代理可以在训练期间多玩大约 4 倍的游戏。

Algorithm 1 deep Q-learning

Initialize replay memory D with a fixed capacity
Initialize action-value function $\hat{q}$ with random weights $\mathbf{w}$
Initialize target action-value function $\hat{q}$ with weights $\mathbf{w}^- = \mathbf{w}$
for episode $m = 1, \ldots, M$ **do**
    Observe initial frame x_1 and preprocess frame to get state s_1
    for time step $t = 1, \ldots, T$ **do**
        Select action $a_t = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg\max_a \hat{q}(s_t, a, \mathbf{w}) & \text{otherwise} \end{cases}$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Preprocess s_t, x_{t+1} to get s_{t+1}, and store transition (s_t, a_t, r_t, s_{t+1}) in D
        Sample uniformly a random minibatch of N transitions (s_j, a_j, r_j, s_{j+1}) from D
        Set $y_j = r_j$ if episode ends at step $j + 1$;
        otherwise set $y_j = r_j + \gamma \max_{a'} \hat{q}(s_{j+1}, a', \mathbf{w}^-)$
        Perform a stochastic gradient descent step on $J(\mathbf{w}) = \frac{1}{N} \sum_{j=1}^{N} (y_j - \hat{q}(s_j, a_j, \mathbf{w}))^2$ w.r.t. parameters $\mathbf{w}$
        Every C steps reset $\mathbf{w}^- = \mathbf{w}$

RMSProp(请参阅 https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf)在[1]中被用于训练尺寸为 32 的 DQN。在训练期间，他们应用了ε − greedy，在前一百万步中将ε从 1.0 线性退火到 0.1，然后固定为 0.1。回放缓冲区用于存储最近的一百万次转换。为了在测试时进行评估，他们使用ε= 0.05 的ε − greedy。

Reducing Bias: Double Deep Q-Network(DDQN)[2]

DQN（算法 1 的第 12 行）中的最大运算符使用相同的网络值来选择和评估动作。此设置使它更有可能选择高估的值，并导致估计的目标值过度乐观。Van Hasselt 等在[2]中也表明，DQN 算法在 Atari 2600 的某些游戏中遭受了严重的高估。为了防止高估并减少偏差，我们可以将 action selection 与 action evaluation 解耦。

回顾 Double Q 学习,通过随机分配转换来更新两个函数中的一个,可以保持和学习两个动作值函数，从而得到两组不同的函数参数，在此表示为 w 和 w'。对于目标计算，一个函数用于选择贪婪动作，另一个函数用于评估其值：

$$y_t^{Double} = r_t + \gamma \widehat{q}(s_{t+1}, \arg\max_{a'} \widehat{q}(s_{t+1}, a', w), w') \quad (8)$$

注意，action selection(argmax)的函数参数是 w，而 action value 是由另一组参数 w'评估的。

通过在计算 target 中将动作选择和动作评估解耦来减少过高估计的想法也可以扩展到 deep Q learning。DQN 架构中的 target network 为第二个 action-value function 提供了候选者，而无需引入其他网络。同样，greedy action 是由参数为 w 的 online network 生成的，但其值是由 target network 使用参数 w−估计的。生成的算法称为 Double DQN [2]，它仅将算法 1 中的第 12 行替换为以下更新目标：

$$y_t^{DoubleDQN} = r_t + \gamma \widehat{q}(s_{t+1}, \arg\max_{a'} \widehat{q}(s_{t+1}, a', w), w') \quad (9)$$

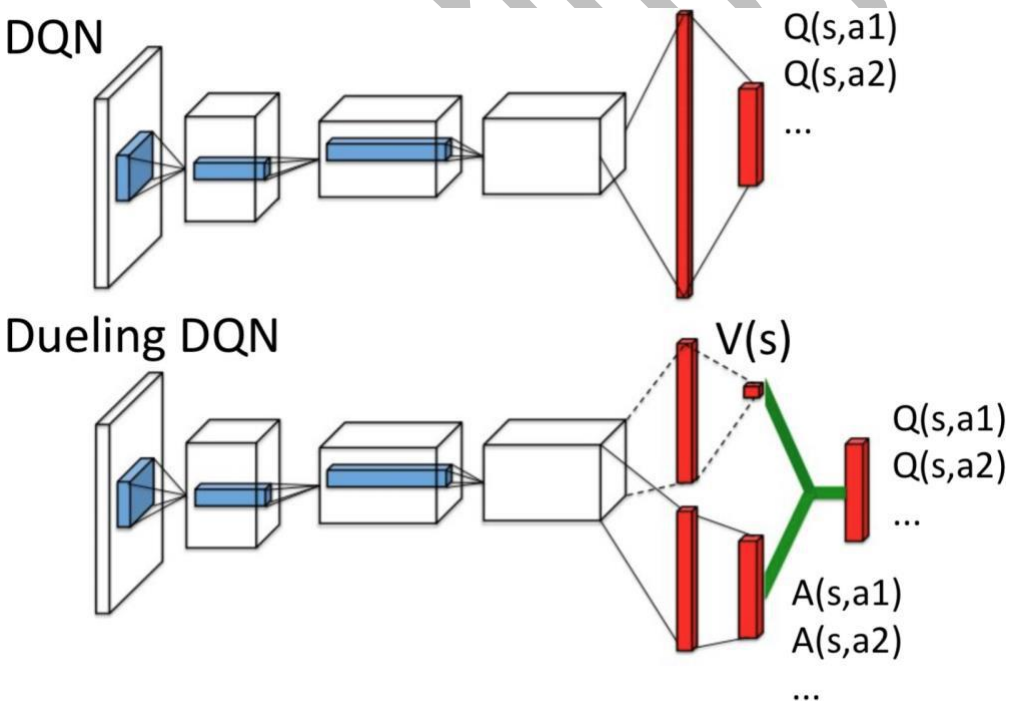

图 3：单流深层 Q 网络（顶部）和对抗 Q 网络（底部）。对抗网络具有两个流，可以分别估计（标量）状态值 V(s) 和每个动作的优势 A(s,a)。绿色输出模块执行公式（13）以合并两个流。两个网络都会为每个动作输出 Q 值.

从 DQN 到 target network 的更新保持不变，并且定义对 online network w 进行备份。DQN 算法的其余部分保持不变。

Decoupling Value and Advantage: Dueling Network[3]

The Dueling Network Architecture

在深入研究 dueling architecture 之前，让我们首先介绍一个重要的变量，即优势函数(advantage function)，该函数将 value 和 Q 函数相关联（假设遵循策略π）：

$$A^{\pi} = Q^{\pi}(s,a) - V^{\pi}(s) \quad (10)$$

复习下我们的值函数公式:

$$V^{\pi}(s) = E_{a\sim\pi(s)}\left[Q^{\pi}(s,a)\right]$$

显然 advantage function 期望收敛为 0:

$$E_{a\sim\pi(s)}\left[A^{\pi}(s,a)\right] = 0$$

直觉上，advantage function 从 Q 函数中减去 state value 以获得对每个动作的相对重要性的度量。

同 DQN 类似, dueling network 也是 DNN 函数逼近器, 用于学习 Q 函数。不同的是, 它通过解耦 value function 和 advantage function 来近似 Q 函数。图 3 说明了 dueling network 结构和 DQN 进行比较。

就像 DQN 中一样，dueling network 的前几层是卷积层。然而，dueling network 不是使用单个全连接的层进行 Q 值估计，而是使用两个全连接层,为方便解释这里称为两个流。一个流用于在给定状态的情况下提供 value function estimate，同时另一个流用于估计每个有效 action 的 advantage function。最后，两条流以产生和近似 Q 函数的方式组合在一起。与 DQN 中一样，网络的输出是 Q 值的向量，每个动作一个。

请注意，由于 dueling network 的输入和最终输出(两个流的合并)与原始 DQN 的相同，因此上面介绍的针对 DQN 和 Double DQN 的训练算法（算法 1）也可以在此处用于训练 dueling architecture。分开的两流设计基于作者的以下观察或直觉：

- 对于许多状态,没有必要估计每个可能的 action choice 的值。在某些状态,动作的选择可能非常重要，但在其他许多状态，动作的选择对接下来发生的事情没有影响。另一方面，对于基于自举的算法(例如 Q 学习),状态值估计对于每个状态都非常重要。
- 确定价值函数所需的功能可能与用于准确估计动作收益的功能不同。

组合两个完全连接的层流以进行 Q 值估计并非易事。实际上，这个聚合模块(图 3 中用绿线显示)需要非常周到的设计，我们将在下一部分中看到它。

Q-value Esimation

根据 advantage function(10)的定义，我们可以得到:

$$Q^{\pi}(s,a) = A^{\pi} + V^{\pi}(s), and\ \mathbb{E}_{a\sim\pi(s)}\left[A^{\pi}(s,a)\right] = 0$$

此外,对于确定性策略(通常在基于值的深层 RL 中使用),$a^{*} = arg\ max_{a'\epsilon A} Q(s,a')$,得出 $Q(s,a^{*}) = V(s)$,因此 $A = (s,a^{*}) = 0$。在这种情况下贪心算法所选择的动作的 advantage function 值为零。

现在思考图 3 中的 dueling network architecture 的函数逼近。让我们将全连接层的一个流的标量输出值函数表示为:$\widehat{v}(s,w,w_v)$，并将另一流的 vector output advantage 表示为 $A(s,a,w,w_A)$。我们在这里使用 w 表示卷积层中的共享参数，并使用w_v和w_A表示完全连接层的两个不同流中的参数。然后，遵循以下定义可能最简单的设计聚合模块的方法:

$$\widehat{q}(s,a,w,w_A,w_v) = \widehat{v}(s,w,w_v) + A(s,a,w,w_A) \quad (11)$$

这种简单设计的主要问题是公式(11)无法识别。在给定$\widehat{q}$的情况下，我们无法准确地恢复$\widehat{v}$和 A，例如，将常数添加到$\widehat{v}$并从 A 减去相同的常数即可得到相同的 Q 值估算值。实践中表现不佳反映了这一无法确定的问题。

为了使 Q 函数可识别，回顾我们在上面讨论的确定性策略案例中，我们可以强制 advantage function 在选定的动作处具有零估计。然后,我们有:

$$\widehat{q}(s,a,w,w_A,w_v) = \widehat{v}(s,w,w_v) + \left[A(s,a,w,w_A) - \max_{a' \sim A} A(s,a',w,w_A)\right] \quad (12)$$

对于确定策略:

$$a^* = arg\max_{a' \sim A} \widehat{q}(s,a',w,w_A,w_v) = arg\max_{a' \sim A} A(s,a',w,w_A)$$

,方程(12)给出:

$$\widehat{q}(s,a^*,w,w_A,w_v) = \widehat{v}(s,w,w_v)$$

因此，流$\widehat{v}$提供了该 value function 的估计值，另一个流 A 生成 advantage estimates。

[3]中的作者还提出了一个替代上诉方案的聚合模块，该模块用均值运算符代替 max:

$$\widehat{q}(s,a,w,w_A,w_v) = \widehat{v}(s,w,w_v) + \left[A(s,a,w,w_A) - \frac{1}{|A|}\sum_{a'} A(s,a',w,w_A)\right] \quad (13)$$

尽管这种设计在某种意义上失去了$\widehat{v}$和 A 的原始语义(original semantics),但作者认为这提高了学习的稳定性:advantages 仅需要和平均值一样快地改变，而不必对最佳动作优势的改变进行补偿。因此，dueling network[3]中的聚合模块是根据方程(13)实现的。在行动时，评估优势流就足以做出决策。

dueling network 的优点在于其有效逼近值函数的能力。当动作数量很大时，与 single-stream Q network 相比，这种优势会增加，并且到 2016 年，dueling network 在 Atari 游戏中获得了最佳的结果