A Algorithm

Algorithm 1 provides pseudo code for RD² on the Atari environment, which learns sub-Q network with jointly learned reward decomposition. Note that RD² can plug in any Q-learning based methods. We found that the second variant of \(\mathcal{L}_{\text{div}}\) works better in Atari. At each time step, we first interact with environments, collect samples in replay buffer (Line 3 to 6). We then train the sub-reward prediction network to predict the total reward with minimal sufficient supporting sub-state (Line 9). We also train the auxiliary prediction network to predict sub-reward \(r_i\) using sub-state \(\hat{s}_j\) (Line 10) to compute \(\mathcal{L}_{\text{div2}}\). After that, we update the mask network \(m_i\) to encourage diversity between sub-states (Line 13).

To train our RL agent, we first perform standard Q-learning using TD error (Line 16) with the full work. As shown in Figure 6, when \(\eta_1 = 0.25e^{-5}\). However, \(K\) could vary depending on the games we choose. Following Castro et al. [2018], we use \(K = 2\). We use a large learning rate (\(\alpha = 10 \times \eta_1\)) to minimize \(\mathcal{L}_{\text{sum}}\). We sweep the learning rate \(\beta, \gamma, \eta_2\) in \(\{0.0, 0.1, 0.01, 0.001, 0.0001, 0.00001\} \times \eta_1\) and finally choose \(\beta = 0.0001 \times \eta_1, \gamma = 0.1 \times \eta_1, \eta_2 = 0.0001 \times \eta_1\). In RD², we update parameters with \(n_{\text{mini}} = 4, n_{\text{div}} = 16, n_{\text{update}} = 4, n_{\text{subq}} = 4\). We use Adam [Kingma and Ba 2014] to optimize all parameters.

B Hyper-parameters

We build our code using the supplied implementation of Castro et al. [2018]. For all experiments we use \(K = 2\). However, \(K\) could vary depending on the games we choose. Following Castro et al. [2018], we use \(\eta_1 = 6.25e^{-5}\). We use a large learning rate (\(\alpha = 10 \times \eta_1\)) to minimize \(\mathcal{L}_{\text{sum}}\). We sweep the learning rate \(\beta, \gamma, \eta_2\) in \(\{0.0, 0.1, 0.01, 0.001, 0.0001, 0.00001\} \times \eta_1\) and finally choose \(\beta = 0.0001 \times \eta_1, \gamma = 0.1 \times \eta_1, \eta_2 = 0.0001 \times \eta_1\). In RD², we update parameters with \(n_{\text{mini}} = 4, n_{\text{div}} = 16, n_{\text{update}} = 4, n_{\text{subq}} = 4\). We use Adam [Kingma and Ba 2014] to optimize all parameters.

C Ablation Study

To investigate the contribution of each loss term in Algorithm 1 we compare three variants of RD²: (1) RD² without \(\mathcal{L}_{\text{sum}}\); (2) RD² without \(\mathcal{L}_{\text{mini}}\); (3) RD² without \(\mathcal{L}_{\text{div2}}\). As shown in Figure 4, when
we drop the $L_{sum}$ term, $RD^2$ is equivalent to learn with randomly decomposed reward. Therefore, the performance deteriorates dramatically. When we drop the diversity encouraging term $L_{div2}$, we get the trivial reward decomposition, which is not helpful to accelerate the training process. Finally, we find that the minimal sufficient regularization term $L_{mini}$ mainly contributes to the later training process.

Figure 6: Ablation study

D Network Architecture

Figure 7 shows the diagram of $RD^2$ to demonstrate the workflow. $r_i$ can then be plugged into any Q-learning algorithm with multiple sub-Q functions. Note that only one of $L_{div1}$ or $L_{div2}$ is required. In our toy experiment, we use $L_{div1}$. In Atari, we use $L_{div2}$.

Figure 7: $RD^2$ work flow.

Figure 8 shows the detailed network architecture. Multiple arrows indicate different network for each of the $K$ reward channels.
Figure 8: Network architecture of RD$^2$. 