RL: Deep
DQN

Marius Lindauer

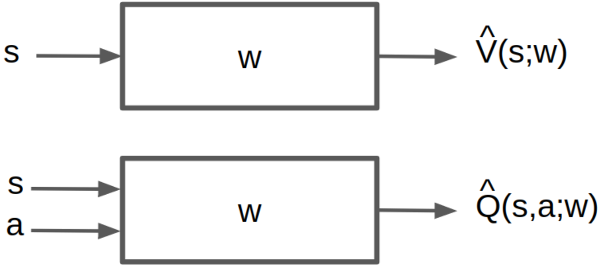
Leibniz Universität Hannover

Automated Machine Learning Hannover

RL with Function Approximation

- Represent state-action value function by Q-network with weights $\mathbf{w}$

$$\hat{Q}(s, a; \mathbf{w}) \approx Q(s, a)$$



Recall: Incremental Model-Free Control Approaches

- Similar to policy evaluation, true state-action value function for a state is unknown and so substitute a target value
- In Monte Carlo methods, use a return G_t as a substitute target

$$\Delta \mathbf{w} = \alpha(G_t - \hat{Q}(s_t, a_t; \mathbf{w}))\nabla_{\mathbf{w}}\hat{Q}(s_t, a_t; \mathbf{w})$$

- For SARSA instead use a TD target $r + \gamma \hat{Q}(s', a'; \mathbf{w})$ which leverages the current function approximations value

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

- For Q-learning instead use a TD target $r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w})$ which leverages the max of the current function approximations value

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

Using these Ideas to do Deep RL in Atari

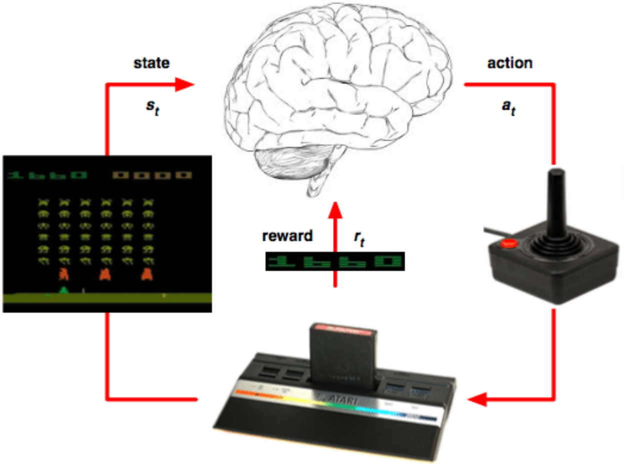
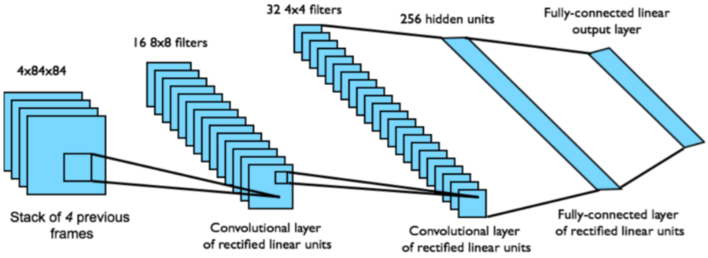


Image by David Silver

Using these Ideas to do Deep RL in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step
- Network architecture and hyperparameters fixed across all games



DQN source code: `sites.google.com/a/deepmind.com/dqn/`

Q-Learning with Value Function Approximation

- Minimize MSE loss by stochastic gradient descent
- Converges to the optimal $Q^*(s,a)$ using table lookup representation
- But Q-learning with VFA can diverge
- Two of the issues causing problems:
  - Correlations between samples violates i.i.d assumption of DNNs
  - Non-stationary targets
- Deep Q-learning (DQN) addresses both of these challenges by
  - Experience replay
  - Fixed Q-targets

DQNs: Replay Buffer

- To help remove correlations, store dataset (called a replay buffer) $\mathcal{D}$ from prior experience
- To perform experience replay, repeat the following:
  1. $(s, a, r, s') \sim \mathcal{D}$: sample experience tuple from the dataset
  2. Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w})$
  3. Use stochastic gradient descent to update the network weights

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

Automated Machine Learning Hannover

DQNs: Replay Buffer

- To help remove correlations, store dataset (called a replay buffer) $\mathcal{D}$ from prior experience
- To perform experience replay, repeat the following:
  1. $(s, a, r, s') \sim \mathcal{D}$: sample experience tuple from the dataset
  2. Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w})$
  3. Use stochastic gradient descent to update the network weights

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

- Remarks:
  - Fixed sized buffer $\rightsquigarrow$ first-in–first-out scheme (as default implementation)
  - heuristic trade-off between performing new episodes and sampling from the replay buffer

DQNs: Replay Buffer

- To help remove correlations, store dataset (called a replay buffer) $\mathcal{D}$ from prior experience
- To perform experience replay, repeat the following:
  1. $(s, a, r, s') \sim \mathcal{D}$: sample experience tuple from the dataset
  2. Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w})$
  3. Use stochastic gradient descent to update the network weights

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

- Remarks:
  - Fixed sized buffer $\leadsto$ first-in–first-out scheme (as default implementation)
  - heuristic trade-off between performing new episodes and sampling from the replay buffer

$\leadsto$ Can treat the target as a scalar, but the weights will get updated on the next round, changing the target value

DQNs: Fixed Q-Targets

- To help improve stability, fix the target weights used in the target calculation for multiple updates
- Target network uses a different set of weights than the weights being updated
- Let parameters $\mathbf{w}^-$ be the set of weights used in the target and $\mathbf{w}$ be the weights that are being updated
- Slight change to computation of target value:
  - $(s, a, r, s') \sim \mathcal{D}$: sample experience tuple from the dataset
  - Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}^-)$
  - Use stochastic gradient descent to update the network weights

$$\Delta \mathbf{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}^-) - \hat{Q}(s, a; \mathbf{w})) \nabla_{\mathbf{w}} \hat{Q}(s, a; \mathbf{w})$$

Automated Machine Learning Hannover

DQNs: Fixed Q-Targets

- To help improve stability, fix the target weights used in the target calculation for multiple updates
- Target network uses a different set of weights than the weights being updated
- Let parameters $\mathbf{w}^-$ be the set of weights used in the target and $\mathbf{w}$ be the weights that are being updated
- Slight change to computation of target value:
  - $(s, a, r, s') \sim \mathcal{D}$: sample experience tuple from the dataset
  - Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}^-)$
  - Use stochastic gradient descent to update the network weights

$$\Delta \mathbf{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}^-) - \hat{Q}(s, a; \mathbf{w}))\nabla_{\mathbf{w}} \hat{Q}(s, a; \mathbf{w})$$

- Remark:
  - Hyperparameter how often you update $\mathbf{w}^-$
  - Trade-off between updating too often ($\leadsto$ instability) and too rarely ($\leadsto$ too old state information)

Automated Machine Learning Hannover

DQN Summary

- DQN uses experience replay and fixed Q-targets
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in reply memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets wrt old, fixed parameters $\mathbf{w}^-$
  - Update $\mathbf{w}^-$ from time to time
- Optimizes MSE between Q-network and Q-learning targets
- Uses stochastic gradient descent