CS 747, Autumn 2022: Lecture 21

Shivaram Kalyanakrishnan

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Autumn 2022
Reinforcement Learning

1. Batch reinforcement learning
   - Experience replay
   - Fitted Q iteration

2. Applications
   - Keepaway soccer
   - Atari 2600 games
1. Batch reinforcement learning
   - Experience replay
   - Fitted Q iteration

2. Applications
   - Keepaway soccer
   - Atari 2600 games
Batch Updates to $\hat{Q}$

- We are back to value function-based learning (with function approximation).

On-line methods such as Q-learning "extract" very little information from each transition; are computationally lightweight. In many applications, samples are more expensive than computation; need to get more out of samples.

Batch RL keeps transitions in memory, performs more computationally-intensive updates.

Batch RL outer loop:

\[ \hat{Q} \leftarrow 0, \quad D \rightarrow \emptyset. \]

Repeat for ever: //Each iteration is a batch.

\[ \pi \leftarrow \epsilon\text{-greedy}(\hat{Q}). \]

Follow $\pi$ for $N$ episodes; gather data $D' = (s_i, a_i, r_i, s_{i+1})$ $L_i = 1$.

$D \leftarrow D \cup D'$.

$\hat{Q} \leftarrow \text{BatchUpdate}(D, \hat{Q})$. // $\hat{Q}$ optional in RHS.

Shivaram Kalyanakrishnan (2022)
Batch Updates to $\hat{Q}$

- We are back to value function-based learning (with function approximation).
- On-line methods such as Q-learning “extract” very little information from each transition; are computationally lightweight.
Batch Updates to $\hat{Q}$

- We are back to value function-based learning (with function approximation).
- On-line methods such as Q-learning “extract” very little information from each transition; are computationally lightweight.
- In many applications, samples are more expensive than computation; need to get more out of samples.
Batch Updates to $\hat{Q}$

- We are back to value function-based learning (with function approximation).
- On-line methods such as Q-learning “extract” very little information from each transition; are computationally lightweight.
- In many applications, samples are more expensive than computation; need to get more out of samples.
- Batch RL keeps transitions in memory, performs more computationally-intensive updates.
Batch Updates to $\hat{Q}$

- We are back to value function-based learning (with function approximation).
- On-line methods such as Q-learning “extract” very little information from each transition; are computationally lightweight.
- In many applications, samples are more expensive than computation; need to get more out of samples.
- **Batch RL** keeps transitions in memory, performs more computationally-intensive updates.

**Batch RL outer loop**

$\hat{Q} \leftarrow 0, \ D \rightarrow \emptyset$.

**Repeat** for ever: //Each iteration is a *batch*.

\[ \pi \leftarrow \epsilon\text{-greedy}(\hat{Q}). \]

Follow $\pi$ for $N$ episodes; gather data $D' = (s_i, a_i, r_i, s_{i+1})_{i=1}^{L}$.

$D \leftarrow D \cup D'$.

$\hat{Q} \leftarrow \text{BatchUpdate}(D, \hat{Q})$. // $\hat{Q}$ optional in RHS.
Experience Replay

- Assume $\hat{Q}$ is function-approximated, say by a neural network.

$$\text{BatchUpdateExperienceReplay}(D, \hat{Q})$$

Repeat $M$ times:
- Pick $(s, a, r, s')$ uniformly at random from $D$.
- Tweak $\hat{Q}$ so that for input $(s, a)$, the output “better-matches” target $r + \gamma \max_{a' \in A} \hat{Q}(s', a')$ (for example by gradient descent).

Return $\hat{Q}$.

- Sometimes $\hat{Q}$ reset/forgotten before the batch update.
- $M$ usually large; hence multiple updates using each sample.
Fitted Q Iteration

- Idea: obtain \( \hat{Q} \) using supervised learning. Wait—labels?

\[
\text{BatchUpdateFittedQIteration}(D)
\]

\[
\hat{Q}_0 \leftarrow 0.
\]

\textbf{For} \( i = 0, 1, \ldots, H - 1: \)

\textbf{For} \( j \in \{1, 2, \ldots, L\}: \) //Create a labeled data set.

\[
x_j \leftarrow \text{FeatureVector}(s_j, a_j).
\]

\[
y_j \leftarrow r_j + \gamma \max_{a \in A} \hat{Q}_i(s_{j+1}, a).
\]

\[
\hat{Q}_{i+1} \leftarrow \text{SupervisedLearning}((x_j, y_j)_{j=1}^L).
\]

\textbf{Return} \( \hat{Q}_H. \)

- Will not diverge if the supervised learning model is an averager (nearest neighbour methods, decision trees, etc.).
Reinforcement Learning

1. Batch reinforcement learning
   - Experience replay
   - Fitted Q iteration

2. Applications
   - Keepaway soccer
   - Atari 2600 games
Keepaway Task, Learning Architecture

- **See video:** [https://www.cs.utexas.edu/~AustinVilla/sim/keepaway/mp4/InitialResults/learn360.mp4](https://www.cs.utexas.edu/~AustinVilla/sim/keepaway/mp4/InitialResults/learn360.mp4).

  - **States:** specified distances, angles between players, play area.
  - **Actions:** hold ball; pass to closer teammate; pass to farther teammate.
  - **Reward:** time between state and next state.
  - **No discounting.**
  - **$\hat{Q}$ approximated by:** (1) tile coding, (2) neural network with 1 hidden layer.
Keepaway Task, Learning Architecture

- **See video**: [https://www.cs.utexas.edu/~AustinVilla/sim/keepaway/mp4/InitialResults/learn360.mp4](https://www.cs.utexas.edu/~AustinVilla/sim/keepaway/mp4/InitialResults/learn360.mp4).

- Only learn policy of keeper with ball.
- **States**: specified distances, angles between players, play area.
- **Actions**: hold ball; pass to closer teammate; pass to farther teammate.
- **Reward**: Time between state and next state.
- No discounting.

^\hat{Q}\text{ approximated by (1) tile coding, (2) neural network with 1 hidden layer.}
Keepaway Task, Learning Architecture

- **See video:** [https://www.cs.utexas.edu/~AustinVilla/sim/keepaway/mp4/InitialResults/learn360.mp4](https://www.cs.utexas.edu/~AustinVilla/sim/keepaway/mp4/InitialResults/learn360.mp4).

- **Only learn policy of keeper with ball.**
- **States:** specified distances, angles between players, play area.
- **Actions:** hold ball; pass to closer teammate; pass to farther teammate.
- **Reward:** Time between state and next state.
- **No discounting.**

- $\hat{Q}$ approximated by (1) tile coding, (2) neural network with 1 hidden layer.
Comparison: On-line vs. Batch RL

Breakout

- Human-level control through deep reinforcement learning.
**Breakout**

- Human-level control through deep reinforcement learning.

See video: [https://www.youtube.com/watch?v=TmPfTpjtdgg](https://www.youtube.com/watch?v=TmPfTpjtdgg).
**Breakout**

- **Human-level control through deep reinforcement learning.**

  See video: https://www.youtube.com/watch?v=TmPfTpjtdgg.
  Observe early, middle, and late stages of training.
Atari 2600 Games: Aggregate Results

From Mnih et al. (2015); for full reference see Slide 9.
Neural Network-based Representation of $Q$

- **Input:** 4 most-recent $84 \times 84$ frames. **Output:** 18 action values.

From Mnih *et al.* (2015); for full reference see Slide 9.
Neural Network-based Representation of $Q$

- **Input:** 4 most-recent $84 \times 84$ frames. **Output:** 18 action values.

From Mnih *et al.* (2015); for full reference see Slide 9.

- Tens of thousands of weights! How to train?
DQN Algorithm

- Batch RL, using **experience replay**.
  - A “mini-batch” of \((s, a, r, s')\) tuples replayed for a few iterations.
  - Q network for providing targets not updated after every atomic update, but still at regular intervals.
DQN Algorithm

- Batch RL, using experience replay.
  - A “mini-batch” of \((s, a, r, s')\) tuples replayed for a few iterations.
  - Q network for providing targets not updated after every atomic update, but still at regular intervals.

- Rewards clipped to \([-1, 1]\).

- **No game-specific** features or hyperparameter-tuning.
DQN Algorithm

- Batch RL, using experience replay.
  - A “mini-batch” of \((s, a, r, s')\) tuples replayed for a few iterations.
  - Q network for providing targets not updated after every atomic update, but still at regular intervals.

- Rewards clipped to \([-1, 1]\).
- **No game-specific** features or hyperparameter-tuning.

- Applied and evaluated on \(\approx 50\) games.
- **Code published**: many implementations now available.
- Results have been improved, **new algorithms** (such as A3C) have emerged.
Summary

- Batch RL motivated by need to
  - conserve samples (by trading off with compute time),
  - handle stability issues with function approximation.
Summary

- Batch RL motivated by need to
  - conserve samples (by trading off with compute time),
  - handle stability issues with function approximation.

- Experience replay most simple, common, effective variant.
Summary

- Batch RL motivated by need to
  - conserve samples (by trading off with compute time),
  - handle stability issues with function approximation.

- Experience replay most simple, common, effective variant.

- Fitted Q iteration also popular; enjoys stability guarantee.
Summary

- Batch RL motivated by need to
  - conserve samples (by trading off with compute time),
  - handle stability issues with function approximation.

- Experience replay most simple, common, effective variant.

- Fitted Q iteration also popular; enjoys stability guarantee.

- Data set $D$ can interpreted as an implicit representation of model.
Summary

- Batch RL motivated by need to
  - conserve samples (by trading off with compute time),
  - handle stability issues with function approximation.

- Experience replay most simple, common, effective variant.

- Fitted Q iteration also popular; enjoys stability guarantee.

- Data set $D$ can interpreted as an implicit representation of model.

- Next class: Model-based methods (again).