# LEARNING TO MANAGE BRIDGES SUBJECT TO SEISMIC HAZARD USING DEEP Q-NETWORKS

*PEER TRANSPORTATION SYSTEMS RESEARCH PROGRAM*

*Principal Investigator:* Jack Baker, Stanford
*Student Investigator:* Gitanjali Bhattacharjee, Stanford
Department of Civil and Environmental Engineering, Stanford University

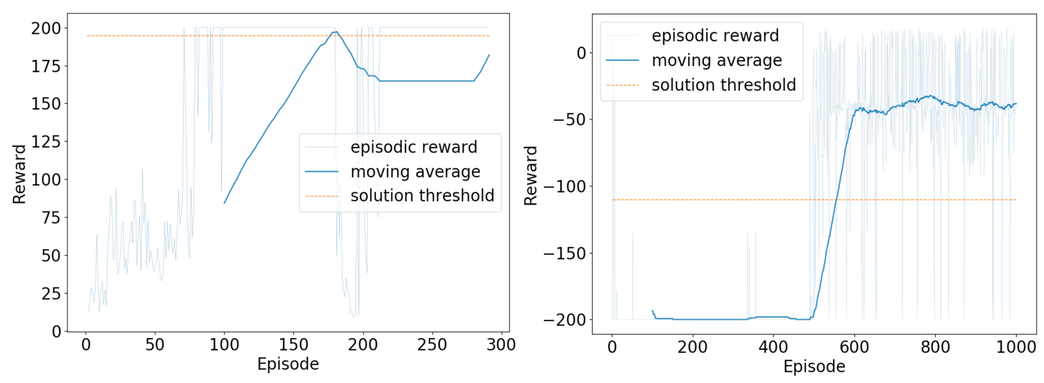
## Learning to manage bridges

This project investigates the application of (double) deep Q-learning to an open problem in civil engineering: the optimal management of a network of bridges subject to seismic hazard. The reinforcement learning agent's goal is to manage each bridge in network such that some network performance objective – for example, total travel time across the network – is met over a planning horizon.

Deep Q-learning (DQN) is a reinforcement learning technique in which a neural network (NN) approximates the action-value function, Q(*s*, *a*), where *s* is a state and *a* is an action [1]. For this application, the function represents the future network-level performance of the system, subject to actions taken to repair or retrofit bridges. Double deep Q-learning (DDQN) partially decouples action selection and evaluation, reducing error in Q-value estimates [2].

## Classic verification problems

We have implemented a DDQN agent and used it to solve the verification problems CartPole-v0 and MountainCar-v0. As shown in the figures below, the agent's performance meets or exceeds the solution threshold for each problem, indicating successful implementation. For CartPole, the agent used an NN with two hidden layers (HLs) of 8 and 4 nodes. For MountainCar, the agent used an NN with three HLs of 256, 128, and 64 nodes. All HLs had rectified linear unit (ReLU) activation functions with He uniform variance scaling initialization; the output layer used a linear activation function.




**The DDQN agent solves OpenAI Gym verification problems: CartPole (left) after 195 episodes and MountainCar (right) after 457 episodes.**

## Deep Q-network inputs and outputs

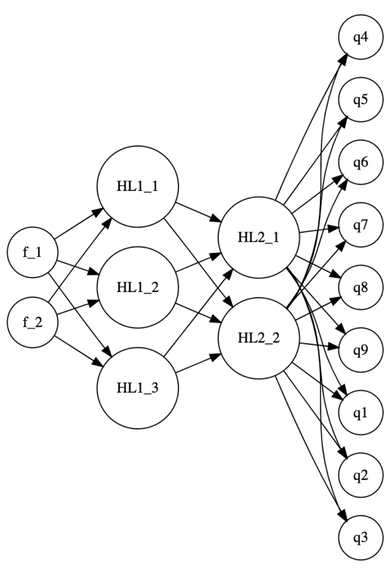
The NN input is a state vector with *b* elements, each a fragility function parameter *f* of a bridge (from CalTrans).

$$\vec{s} = [f_1, ..., f_b]$$

Unless damaged, a bridge has a non-zero *f*. Larger values of *f* indicate greater resistance to ground shaking. The agent can do nothing to, retrofit (double the *f* of), or repair (restore the original *f* of) each bridge. The size of the action space $|A| = 3^b$, so the NN outputs a vector with $3^b$ elements, each corresponding to a predicted Q(*s*, *a*).

$$\vec{Q} = [Q(s, a_1), ... Q(s, a_{|A|})]$$

The figure below diagrams a NN with two HLs (of 3 and 2 nodes, respectively) for a problem involving two bridges ($b = 2$) and three actions at each bridge ($|A| = 9$). Each of the 9 output nodes corresponds to a predicted Q(*s*, *a*) for the input state *s* and a possible action for the system. For this project, we consider three bridges; thus $b = 3$ and $|A| = 27$.




In general, an agent chooses an action to take at each time-step of an episode. In this context, a time-step is one year. At the beginning of each time-step, the agent has the chance to repair or retrofit bridges. After the agent has chosen an action, an earthquake (from the UCERF2 catalog) may occur. Bridges may sustain damage due to earthquakes. Here, we consider bridges with at least extensive damage to be "damaged".

## One-step episode returns

In a multi-step episode in Q-learning,

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(s', a')$$

where $r(s, a)$ is a rewards function, γ is a discount factor with value [0, 1], and *s'* and *a'* are the next state and next action, respectively. For a one-step episode, this reduces to

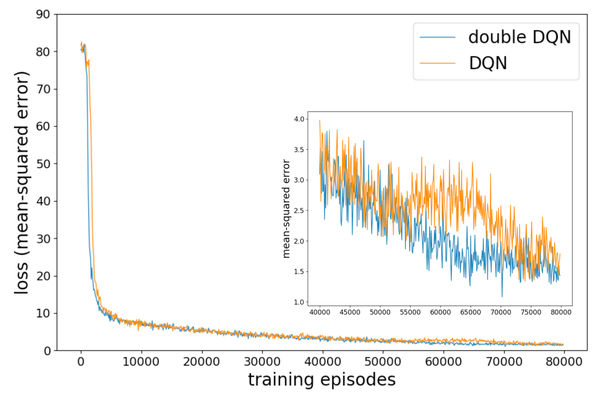
$$Q(s, a) = r(s, a)$$

which makes evaluation of an NN's predictive power straightforward. Since there are no standard NN architectures for this problem, multiple NNs were tested on their ability to predict one-step episode returns.

## Predicting one-step episode returns

The NNs' predictive power was measured by the loss recorded on a batch of 64 one-step episodes randomly selected from memory and periodically evaluated during training. The following rewards function was used as a proxy for system performance, with $b = 3$ bridges in the network. All actions were free. This function will be replaced with a traffic model and real costs in future iterations.

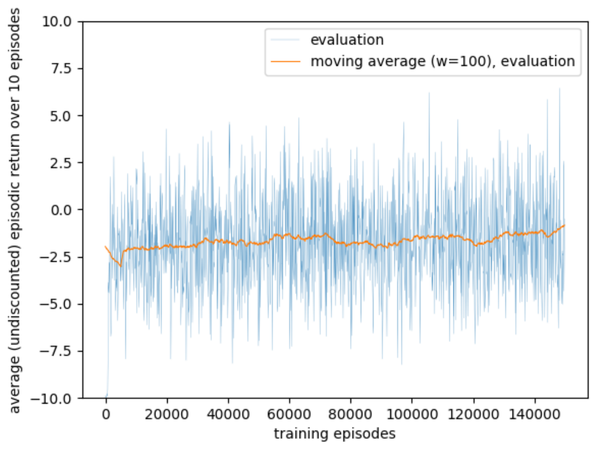
$$r(s, a) = \frac{1}{b} \sum_{i=1}^{b} F_i, \text{ where } F_i = \begin{cases} 1 & \text{if bridge } i \text{ is undamaged} \\ -1 & \text{if bridge } i \text{ is damaged} \end{cases}$$




**Loss of one-step return predictions over time, with zoom inset.**

Multiple NNs were tested, with two to five HLs, each with five to 500 hidden units. All HLs had ReLU activation functions with He uniform variance scaling initialization, while the output layer used a linear activation function. All NNs used the Adam optimization algorithm and a mean-squared-error loss metric. The best-performing NN had two HLs, each with 300 nodes. The DDQN agent outperformed the DQN agent, with a minimum loss of 1.1 after 80000 episodes, compared to 2.7 for the DQN agent.

## Longer-term results




The undiscounted return – that is, the sum of the reward, $r(s,a)$, at each time-step – for a ten-year episode ranges from -10 to 10. The agent was evaluated every 100 training episodes; its average return over 10 decade-long evaluation episodes with random starting states is plotted at left. Over 150000 training episodes, the DDQN agent achieved a maximum return of 6.4 during evaluation.

## Discussion

The DDQN agent learns to repair and retrofit bridges, improving its average episodic return with training. However, the improvement is slight. As bridge repairs and retrofits are free under the above reward function used for testing, the agent was expected to achieve a near-optimal return during evaluation, but did not. As increasing the NN depth from 2 to 5 HLs did not improve results over 50k training episodes, under-fitting may not be the primary concern. The agent trained for a maximum of 150000 episodes in an action space of size 27. Mnih et al. trained their agent for 10 million episodes, though it acted in much smaller action spaces (of size 4 to 18) [1]. This suggests that training may have been too short. Immediate future work will include more closely examining how the agent policy changes with training. The risk function will be refined to represent realistic network performance, costs, more bridges, and a longer planning time horizon.

### References

1. Mnih, Volodymyr, et al. Playing Atari with deep reinforcement learning. *NIPS Deep Learning Workshop*, 2013.
2. van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double Q-learning. arXiv preprint arXiv:1509.06461, 2015.

*This project was made possible with support from:*

PACIFIC EARTHQUAKE ENGINEERING RESEARCH CENTER
UC Berkeley • Caltech • OSU • Stanford • UC Davis • UC Irvine • UC Los Angeles • UC San Diego • UNR • USC • U Washington
peer.berkeley.edu