MDP

Q Learning. State the authors are using a deep network to approximate the Q(s,a) function.

Mention success of TD-Backgammon

RL can see actions yield result after many timesteps. Trial & error, punishment & reward type system.

Because it is impossible to fully understand current situation from only the current screen.

(Not sure of direction or speed …etc.)

Q(s,a)

Richard Bellman - Bellman Equation – foundation of dynamic programming

Q-Learning

Q-Network is a NN function approximator to the Q-Value function. DQN is the whole algorithm on the next slide.

Replace value iteration with non-linear function approximator (NN). Theta are parameters (weights) of network

Mention this is model-free RL because we are not using the rest of the model just approximating q-values

Solves RL problem from the emulator without explicitly constructing an estimate of the emulator.

Off-policy – updates the Q-values using next state and greedy action. (on-Policy – updates Q-Values using next state Q-value and current policy’s action. )

Learns about the greedy strategy (to select the action to maximise q-value)

That is it selects the maximising action a = max Q(s,a,theta) with probability 1-e & selects a random action with probability e

Learning can be achieved

The targets for learning at the current iteration depend on the network weights at the previous iteration this is different to supervised learning where targets for learning are fixed. Target for learning is the reward plus max of the q-value as suggested by the previous state of the network

Deep nn can extract features to learn better representations that can be achieved through handcrafting features.

TD-Gammon architecture is some inspiration because it used NN to estimate Value function.

Authors use Experience Replay

Perform Q-learning updates to mini-batches in the inner loop of the algorithm.

Final cropping step only needed because authors use GPU implementation that requires square inputs. (Disclaimer! my diagram does not visualise cropping!)

Advantage is ability to compute Q-values for all possible actions for a state with only one forward pass through the network.

7 Games: Beam Rider, Breakout, Enduro, Pong, Q\*bert, Seaquest, Space Invaders

Positive rewards goto 1, negative rewards goto -1 & zero rewards remain at zero.

This limits the scale of the error derivatives making it easier to same learning rate hyperparameter across games.

Note: this could limit agent performance since there is no differentiation between rewards of different magnitude.

Frame skipping technique, agent sees every kth frame with last action being repeated on frames it doesn’t see.

Easier skip emulator forward than select action for each frame, this allows k times more games to be played.

K=4 for games except for Space Invaders where k=3 (to make the lasers visible to the agent)

Since the evaluation metric (loss function) is the total reward collected the authors compute it over training.

However it doesn’t give the impression of learning progress because the signal is noisy. (because small, weight changes result in large changes to the game states visited)

Q-Value is more stable metric (right), this is reward obtained from current state.

The values for all 4 plots are averaged over a fixed set of states collected at the beginning training. The Q-Value is taken from each state by maximising over its actions.

The smooth improvement to predicted Q-value shows the method is able to train large neural networks using an RL signal.

For the learned methods we report average score obtained following an e-greed policy with e=0.05

Two past s-o-a use hand engineered feature sets.

DQN is better despite no prior knowledge about the inputs/representations (Notable convolutional nets as feature extractors is novel for atari)

NHEAT HERE

Two comparisons of Nheat (evolutionary policy search algorithm)

Better than human expert on Breakout, Enduro, Pong and close on Beam Rider

Far from human performance on Q\*bert, Seaquest and Space Invaders.