DQN and Atari Games (Intro to Deep RL)

Ron Parr

CSCI 2951-F

Brown University

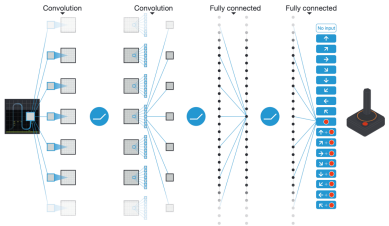


Illustration from Mnih et al. (2015)

How It Started: Learning to play Backgammon

- Neurogammon developed in 1989 using supervised learning
  - Trained NN on expert human moves
  - Played at level of intermediate human player

- TD-gammon developed in 1992 using RL
  - Neural network value function approximation
  - TD sufficient (known model)
  - Using raw board positions, learned to play as well as neurogammon
  - Tesauro added carefully selected features to the network
  - Then had it play 1 million games played against self
  - Comparable performance to best human players

About Atari Games

- Atari 2600 was an early generation video game that accepted ROM cartridges enabling a variety of games on the same hardware – arguably the first widely successful system of this type
- Had very limited storage, graphics, sound, computation
  - 160x192 resolution
  - 128 bytes of RAM
  - 4K ROM cartridges
  - 1.2 Mhz processor (MOS 6507)
- Simple controller with 1 button, 8-directional joystick

- In short, it was the equivalent of an Xbox, Playstation or Switch in RP's youth

How awful is 6807 programming?

```
10 MPR=$C0 ;MULTIPLIER
20 MPD1=$C1 ;MULTIPLICAND
30 MPD2=$C2 ;NEW MULTIPLICAND AFTER 8 SHIFTS
40 PRODL=$C3 ;LOW BYTE OF PRODUCT
50 PRODH=$C4 ;HIGH BYTE OF PRODUCT
60 ;
70 *=$0600
80 ;
85 ;THESE ARE THE NUMBERS WE WILL MULTIPLY
87 ;
90  LDA #250
100  STA MPR
110  LDA #2
120  STA MPD1
130 ;
140 MULT CLD
150  CLC
160  LDA #0 ;CLEAR ACCUMULATOR
170  STA MPD2 ;CLEAR ADDRESS FOR SHIFTED MULTIPLICAND
180  STA PRODL ;CLEAR LOW BYTE OF PRODUCT ADDRESS
190  STA PRODH ;CLEAR HIGH BYTE OF PRODUCT ADDRESS
200  LDX #8 ;WE WILL USE THE X REGISTER AS A COUNTER
210 LOOP LSR MPR ;SHIFT MULTIPLIER RIGHT; LSB DROPS INTO CARRY BIT
220  BCC NOADD ;TEST CARRY BIT; IF ZERO, BRANCH TO NOADD
230  CLC
240  LDA PRODL
250  ADC MPD1 ;ADD LOW BYTE OF PRODUCT TO MULTIPLICAND
260  STA PRODL ;RESULT IS NEW LOW BYTE OF PRODUCT
270  LDA PRODH ;LOAD ACCUMULATOR WITH HIGH BYTE OF PRODUCT
280  ADC MPD2 ;ADD HIGH PART OF MULTIPLICAND
290  STA PRODH ;RESULT IS NEW HIGH BYTE OF PRODUCT
300  NOADD ASL MPD1 ;SHIFT MULIPLICAND LEFT; BIT 7 DROPS INTO CARRY
310  ROL MPD2 ;ROTATE CARRY BIT INTO BIT 7 OF MPD2
320  DEX ;DECREMENT CONTENTS OF X REGISTER
330  BNE LOOP ;IF RESULT ISN'T ZERO, JUMP BACK TO LOOP
340  RTS
350  .END
```

This is the assembly code for multiply two numbers 😱

(not a primitive operation in 6502/6507 assembler)

Source: https://www.atariarchives.org/roots/chapter_10.php

Why Atari as an RL Benchmark?

- Easy(ish) to simulate (See MinAtar for another approach)
- Widely available
- Small(ish) discrete action space
- Large number of games possible in a common platform
- Diversity in types of games, including many that required somewhat long term behavior
- No difficult object recognition problems involved (graphics too crude)
- Not obviously easy

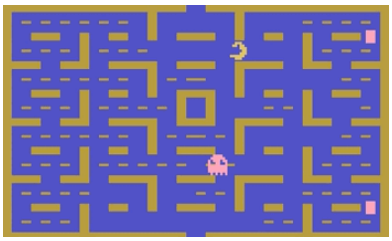
Some challenges

- Single frame is not a Markovian state (partial solution: stack frames)
- Games designed for human time scale responses, ***not*** for changing actions every 1/60 second (solution: make actions sticky)
- Flicker – some objects appeared only in odd or even frames
  - See, e.g., the ghosts in Pac-Man
  - Partial solution: input is max over two adjacent frames

Some games

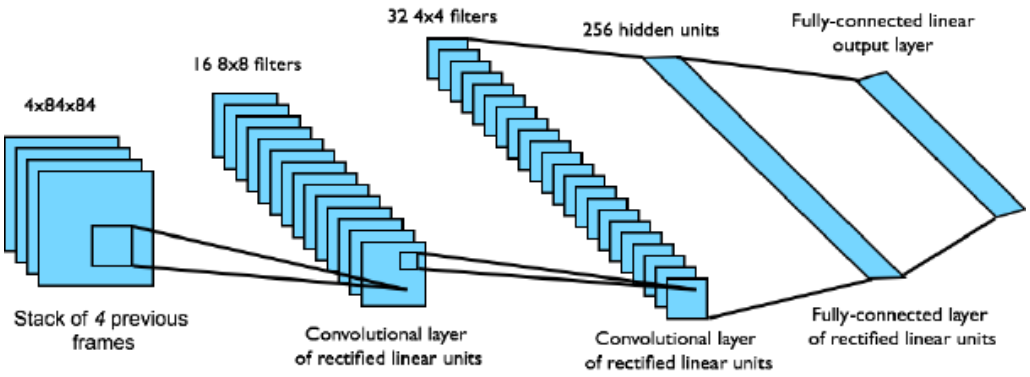
Breakout (Wikipedia)

Space Invaders
(thegameroom.fandom.com)



Pac-Man
(retrogames.cz)

DQN Architecture



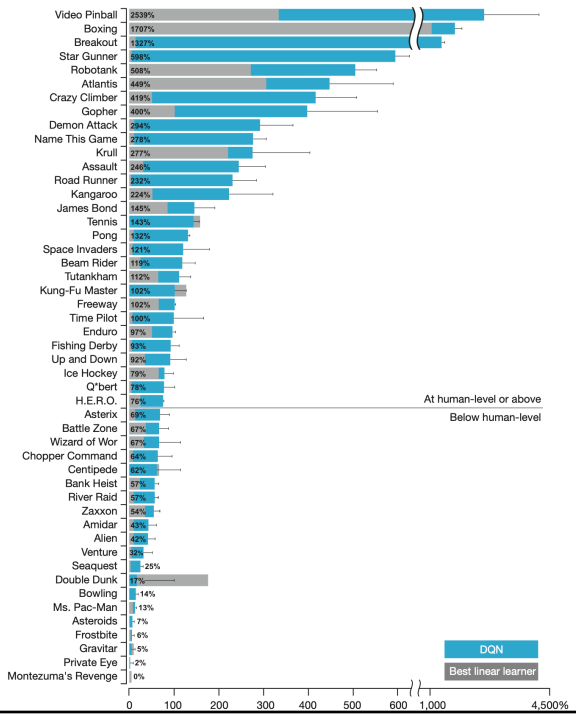
From: https://www.davidsilver.uk/wp-content/uploads/2020/03/deep_rl_compressed.pdf

Changes in training

- Used experience replay (replay buffer)
  - Old technique
  - Slightly closer to batch/fitted Q-iteration
- Used a “Target Q-network”
  - Two copies of neural network
  - Use ”target” network on RHS of Bellman equation
  - After a batch of training, copy over newly learned network to target (another step towards fitted Q-iteration)
- Clip rewards to keep NN gradients from having a wide range

Results

From DQN paper:



Lessons learned

- From TD-Gammon to DQN **surprisingly little as changed**
  - Still no stability or performance guarantees despite changes
  - Training still requires massive amounts of data
  - Convnets, small changes in training make a big difference (as in deep nets)
- Yet everything has changed
  - After years of frustration in applying RL to hard problems, now people want to apply RL to everything
  - Harder games
  - Power management in data centers
  - Robotic control

Unsatisfying aspects of DQN/Atari success

- No high level knowledge (all new games learned from scratch)
- Training time is quite large (50M frames)
- Solutions lack robustness
  (adding irrelevant "distractor" graphics can cause strange behavior)
- Some evidence that solutions may be partly memorized
  (poor generalization)