Teaching Agents with Deep Apprenticeship Learning

Thesis Defense

By: Amar Bhatt

Computer Engineering BS/MS

5th Year

Advisor: Dr. Raymond Ptucha

Thesis Committee Members

Dr. Raymond Ptucha (Advisor), Assistant Professor - Computer Engineering

Dr. Ferat Sahin, Professor - Electrical Engineering

Dr. Iris Asllani, Assistant Professor - Biomedical Engineering

Dr. Christopher Kanan, Assistant Professor - Imaging Science

Professor Louis Beato, Lecturer - Computer Engineering

Agenda

- Reinforcement Learning
- Apprenticeship Learning
- Problem Statement
- Novel Contributions
- Algorithms and Methodologies
- Test Environment
- Results and Conclusion

Agenda

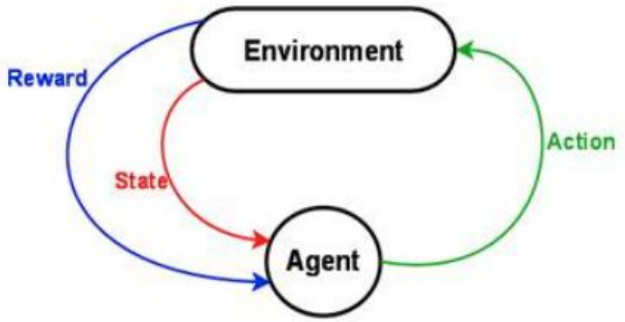
- **Reinforcement Learning**
- Apprenticeship Learning
- Problem Statement
- Novel Contributions
- Algorithms and Methodologies
- Test Environment
- Results and Conclusion

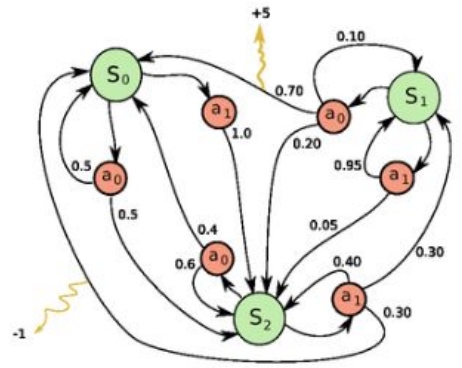
What is Reinforcement Learning?

- Living creatures learn through trials of reward and punishment called Reinforcement Learning
- Solves Markov Decision Process (MDPs) problems
  - The probability of going to the next state depends only on the current state and action, but not preceding states and actions
- Unlike supervised learning it does not require ground truth data

How does it work?

- An agent (car, robot, system) is situated in some environment at some state
  - State ⟶ location of agent, location of obstacles, goal, time, health, etc.
  - Environment ⟶ maze, workspace, field, road, etc.
- An agent can perform some known actions
  - Actions ⟶ move, throw, etc.
  - Actions result in some reward
  - Lead to another state





https://www.nervanasys.com/wp-content/uploads/2015/12/Screen-Shot-2015-12-21-at-12.01.04-PM.png

How does it work?

- Set of states ($s \in S$) with a set of actions ($a \in A$)
- Try to learn a policy (π) for state transition
- One iteration of a complete cycle, game, etc $\rightarrow$ Episode
- Each episode ends with a terminal state $\rightarrow$ end of game, goal, etc.
- The agent must go through many episodes to learn

Rewards

- Immediate reward is received from the current state-action pair
- Future rewards (from future states) are taken into account as well
- Total reward from a state ($\boldsymbol{R_T}$) is the sum of rewards ($\boldsymbol{r}$) received from each state-action pair in a given episode
- Usually future rewards are discounted by γ (0-1):
  - $R_T = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{n-t} r_n$

Tambet Matiisen. Guest post (part i): Demystifying deep reinforcement learning - nervana, Dec 2015.

Q-Learning

- The most popular form of Reinforcement Learning
- Based on Temporal Difference Learning
- Uses estimated equations to learn an environment over numerous iterations

Q-Learning - How does it work?

- For each state-action pair ***(s,a)*** there exists a Q-value that represents the maximum discounted future reward (***Q(s,a)***)
  - The *Q(s,a)* value indicates the maximum reward received when starting at state ***s*** and performing action ***a*** and continuing optimally from that point
  - The Q-values and Rewards are represented by tables of states (rows) and actions (columns)
- The goal of Q-learning is to learn Q-values so that the policy (***π***) is:

$$\pi(s) = max_a\{Q(s,a)\}$$

- One problem
  - How can we estimate the final reward if we only know the current state and action?

Q-Learning - Bellman Equations

- Using an approximate and only looking one state ahead, the Q-values for state-action pairs can be estimated
- Over several iterations, these estimations will become truth
- The current Q-value of state ***s*** and action ***a*** is based on the Q of the next state (***s'***) and next best action (***a'***)

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

- Iteratively approximate Q-function until the updates for $Q(s,a)$ are close to 0

$$Q(s,a)_{new} \leftarrow Q(s,a) + \alpha * [r(s,a) + \gamma * max_{a'}\{Q(s',a')\} - Q(s,a)]$$

for $e \in$ number of episodes **do**

Pick random state *s*

while *s* is not a terminal state **do**

Choose action *a* from *s* using a policy from *Q*

Take action *a*, observe reward *r* and next state *s'*

Update Q-values as follows:

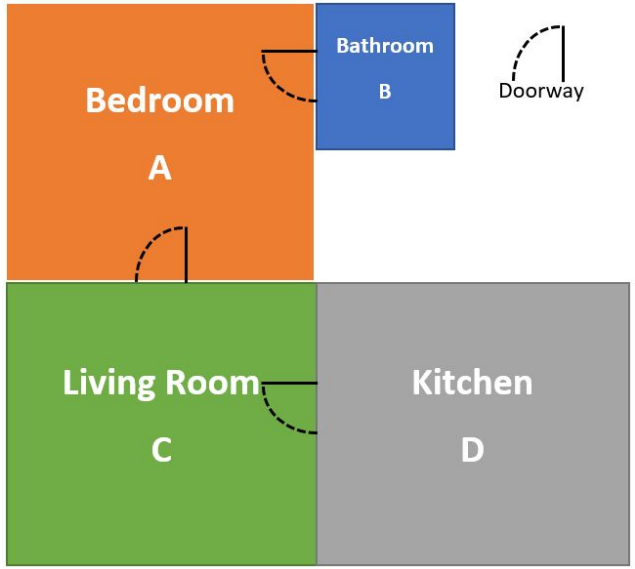
$$Q(s,a)_{new} \leftarrow Q(s,a) + \alpha * [r(s,a) + \gamma * max_{a'}\{Q(s',a')\} - Q(s,a)]$$

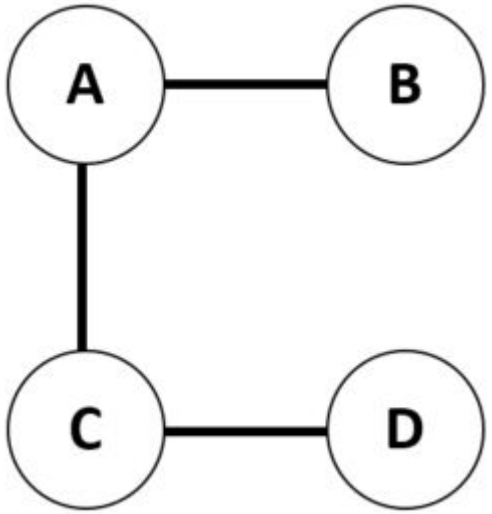
$s \leftarrow s'$

Tim Eden, Anthony Knittel, and Raphael van Uffelen. Reinforcement learning. http://www.cse.unsw.edu.au/~cs9417ml/RL1/index.html. Accessed: 2015-11-22.

Q-Learning - An Example

- 4-room house
- Goal: Get to the kitchen (+100)
- Actions: Move through any door





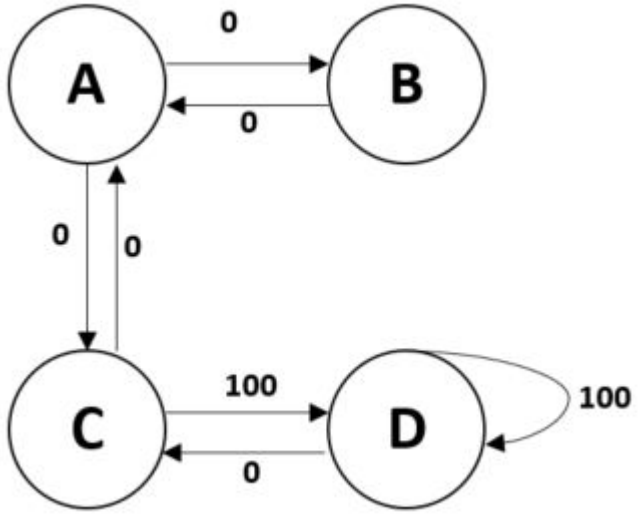
Q-Learning - An Example

- Set-up Q-table

Q^0	Possible next state of agent				
		A	B	C	D
Current state of agent	A	0	0	0	0
	B	0	0	0	0
	C	0	0	0	0
	D	0	0	0	0

- Set-up Reward table

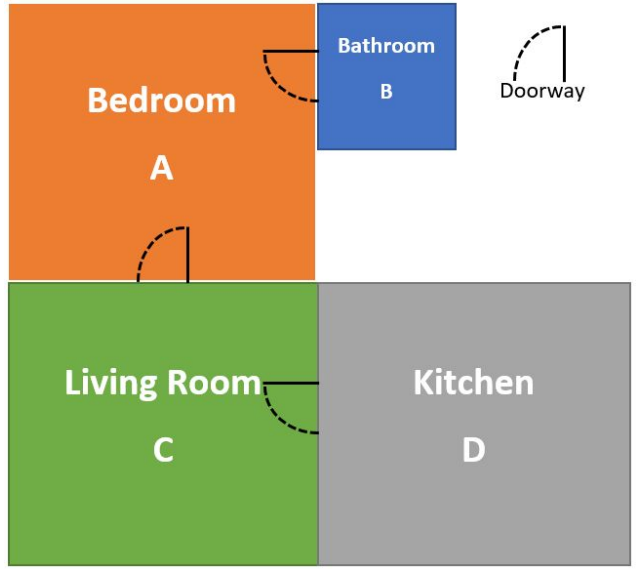
R	Possible next state of agent				
		A	B	C	D
Current state of agent	A	-	0	0	-
	B	0	-	-	-
	C	0	-	-	100
	D	-	-	0	100



Q-Learning - An Example

R	Possible next state of agent			
Current state of agent	A	B	C	D
A	-	0	0	-
B	0	-	-	-
C	0	-	-	100
D	-	-	0	100

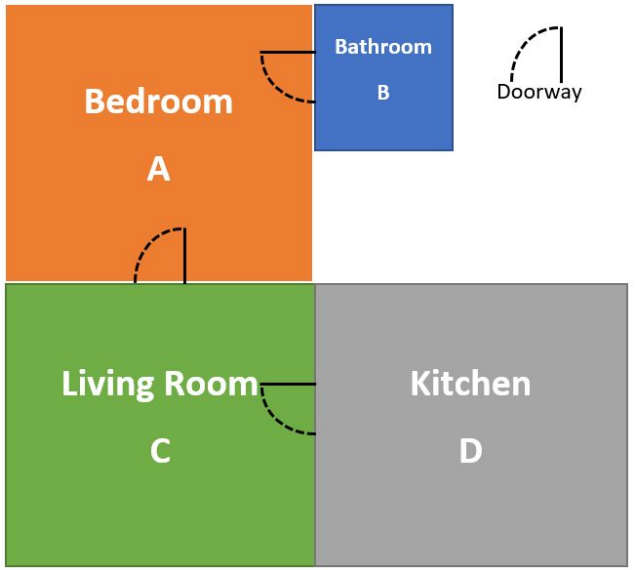
- $\gamma = 0.9$
- Iteration 1:
  - Starts in Bedroom (A) transitions to Living Room (C)
    $Q(A,C) = R(A,C) + \gamma * max_a Q(C, a) = 0 + 0.9 * 0 = \mathbf{0}$
  - From Living Room (C) transitions to Kitchen (D)
    $Q(C,D) = R(C,D) + \gamma * max_a Q(D,a) = 100 + 0.9 * 0 = \mathbf{100}$
  - From Kitchen (D) transitions to Kitchen (D)
    $Q(D,D) = R(D,D) + \gamma * max_a Q(D,a) = 100 + 0.9 * 0 = \mathbf{100}$



Q^1	Possible next state of agent				
Current state of agent	A	B	C	D	
A	0	0	0	0	
B	0	0	0	0	
C	0	0	0	100	
D	0	0	0	100	

Q-Learning - An Example

- Iteration 2:
  - Starts in Bathroom (B) transitions to Bedroom (A)
    $Q(B,A) = R(B,A) + \gamma * max_a Q(A, a) = 0 + 0.9 * 0 = \mathbf{0}$
  - From Bedroom (A) transitions to Living Room (C)
    $Q(A,C) = R(A,C) + \gamma * max_a Q(C,a) = 0 + 0.9 * 100 = \mathbf{90}$
  - From Living Room (C) transitions to Kitchen (D)
    $Q(C,D) = R(C,D) + \gamma * max_a Q(D,a) = 100 + 0.9 * 100 = \mathbf{190}$
  - From Kitchen (D) transitions to Kitchen (D)
    $Q(D,D) = R(D,D) + \gamma * max_a Q(D,a) = 100 + 0.9 * 100 = \mathbf{190}$

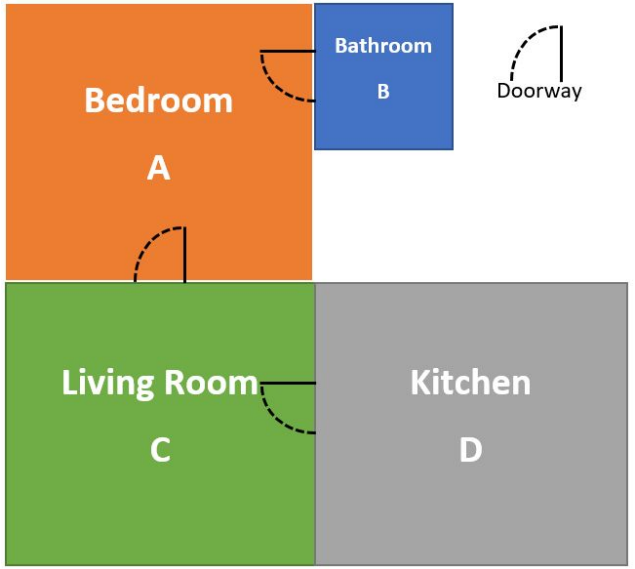


Q^2	Possible next state of agent			
Current state of agent	A	B	C	D
A	0	0	**90**	0
B	**0**	0	0	0
C	0	0	0	**190**
D	0	0	0	**190**

Q-Learning - An Example

- Iteration 3:
  - Starts in Bathroom (B) transitions to Bedroom (A)
    $Q(B,A) = R(B,A) + \gamma * max_a Q(A, a) = 0 + 0.9 * 90 =$ ***81***
- …

Q^3	Possible next state of agent				
Current state of agent		A	B	C	D
	A	0	0	90	0
	B	81	0	0	0
	C	0	0	0	190
	D	0	0	0	190

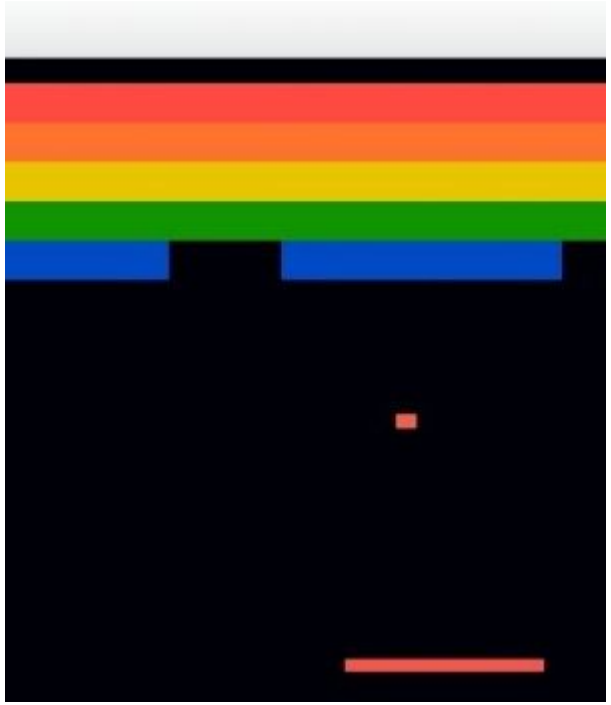


Q-Learning - Extra Considerations

- Exploration vs. Exploitation
  - Exploration → Random actions to discover new paths
  - Exploitation → Do what works
- How do we determine Exploration vs. Exploitation?
  - ε-greedy
    - With probability $\boldsymbol{\varepsilon}$ choose a random action, otherwise choose a learned action
- Need to know reward structure of your environment
- Large state spaces and large action sets make it so Q-learning cannot scale

Q-Networks

- Utilizing Neural Networks to replace the Q-table
  - Allows for large state-action spaces
  - 84x84 grayscale (0-255) image (state space) would result in $256^{7,056}$ possible states
  - Good for feature learning
- Using Deep Convolutional Neural Networks allows for learning from raw pixel data

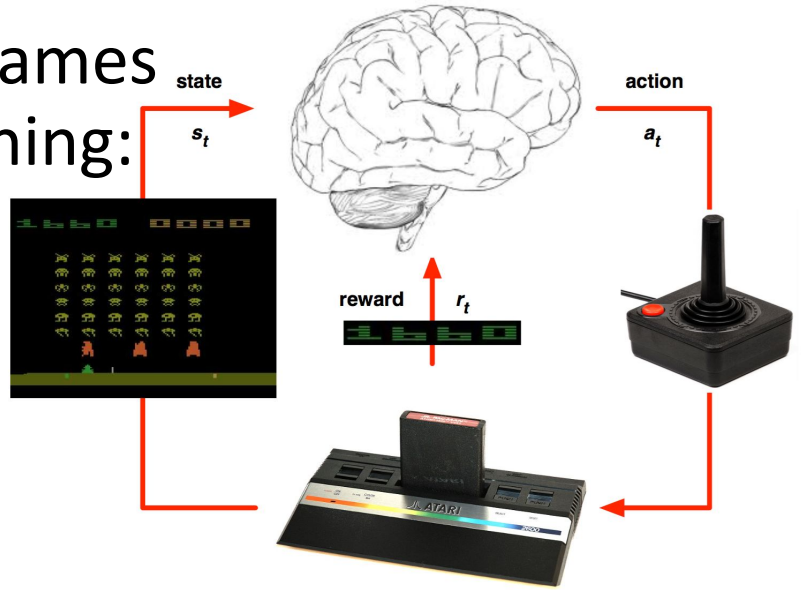


http://www.atari-breakout.com/wp-content/uploads/2015/07/Atari-Breakout-Google_610x350-300x350.jpg

Related Works

Deep Q-Network (Google DeepMind)

- Google DeepMind tested their DQN with Atari games
- Two significant changes from traditional Q-Learning:
  - Experience Replay
    - Storing agent memory <*s,a,s',r*> in limited replay buffer
    - Removes correlation
  - Target Q-Network used for calculations
    - Q-updates can be very large and tend to diverge
    - Brings stability to the network
    - Separate network entirely updated in intervals
- $J(Q) = \left(\left[R(s,a) + \gamma * max_{a'}\{Q(s',a')\}\right] - Q(s,a)\right)^2$
  - Feed-forward to get *Q(s,a)* and *Q(s',a')* values



https://keon.io/images/deep-q-learning/rl.png

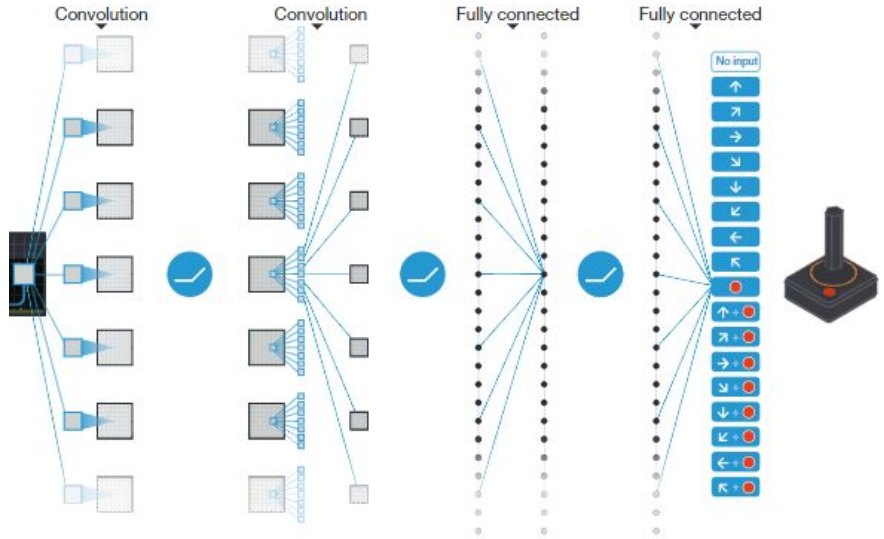
Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, February 2015.

Deep Q-Network (DQN) Architecture

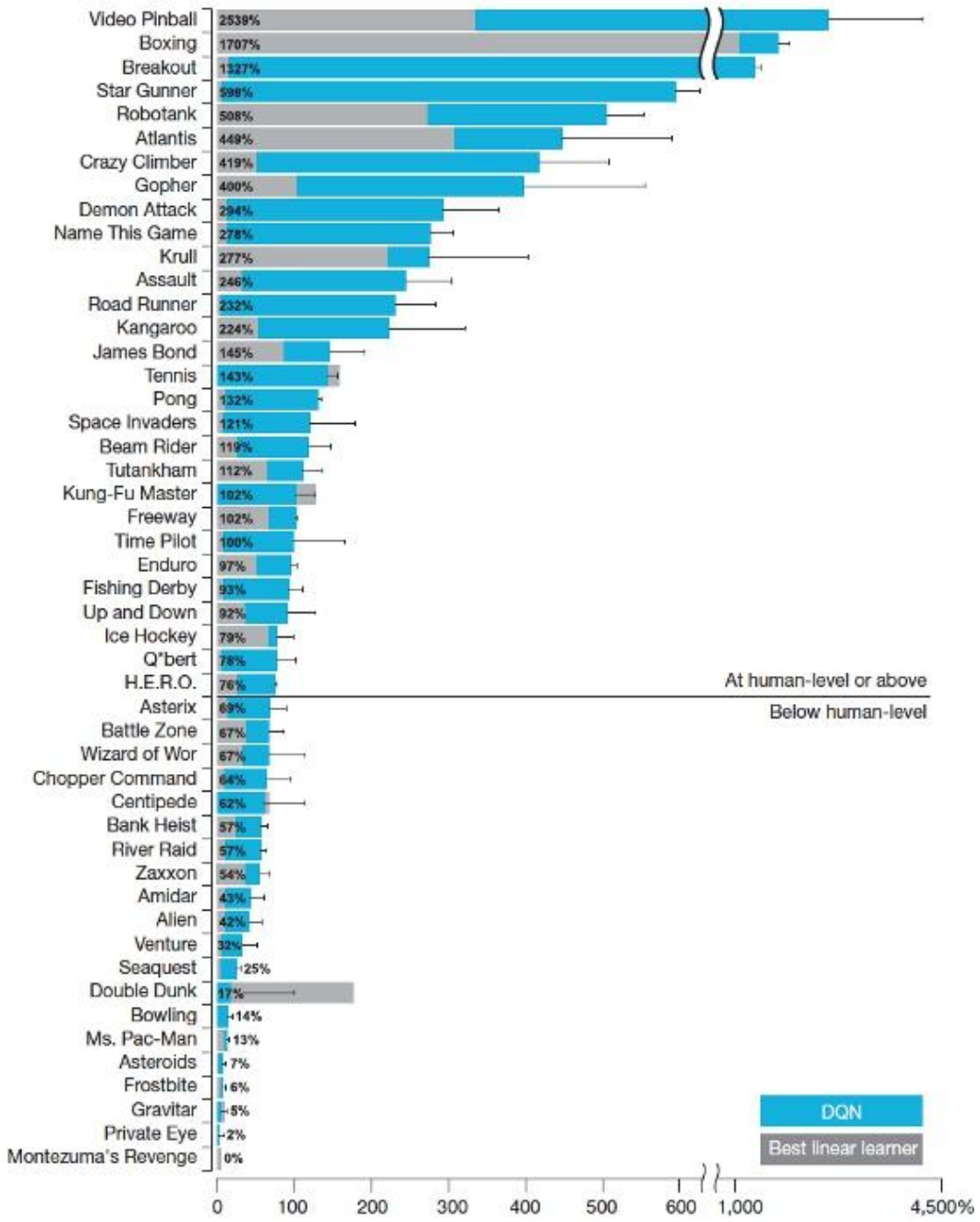
- Input of 84x84x4
- Output of 18 actions

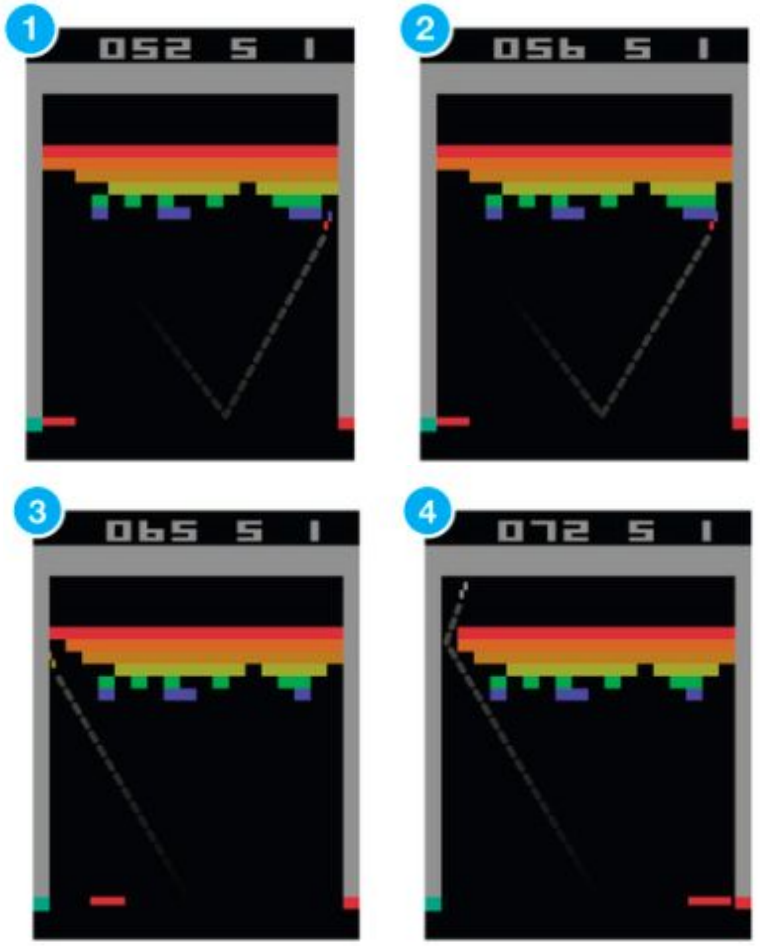
Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8x8	4	32	ReLU	20x20x32
conv2	20x20x32	4x4	2	64	ReLU	9x9x64
conv3	9x9x64	3x3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18

https://www.nervanasys.com/wp-content/uploads/2015/12/Screen-Shot-2015-12-21-at-11.23.28-AM.png



Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, February 2015.





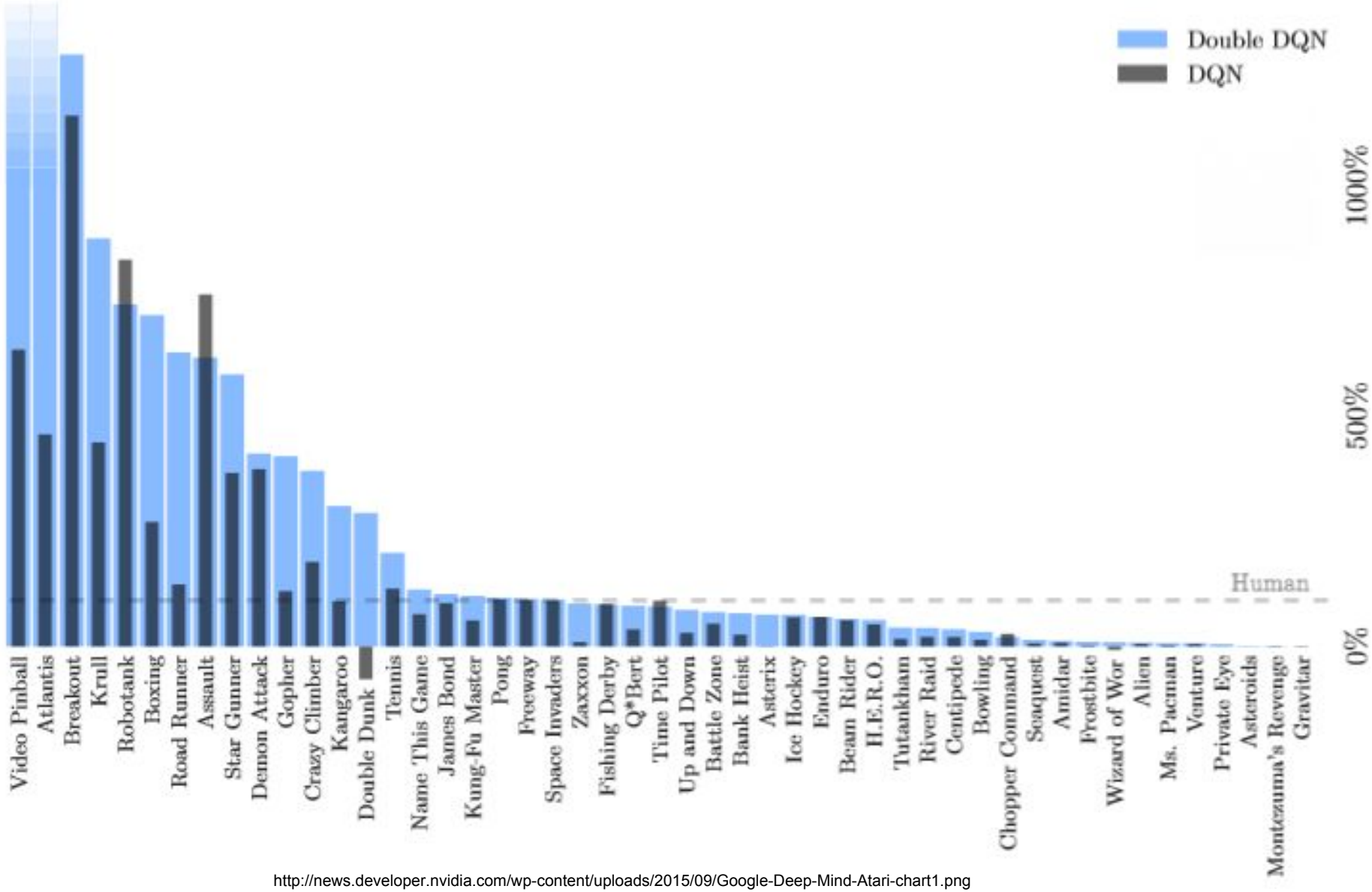
https://adeshpande3.github.io/Deep-Learning-Research-Review-Week-2-Reinforcement-Learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, February 2015.

Double Deep Q-Networks

- Modification to DQNs, uses same architecture
- Uses the Q-values from the separate target Q network
  - Choose action ***a'*** from primary network
    - $max_{a'}\{Q(s',a')\}$
    - Q-values in primary network are grossly overestimated and inaccurate, but the action selection is still valid
  - Use target Q-network to generate Q-value for ***s'*** using ***a'***
    - $R(s,a) + \gamma * Q^{target}(s',a')$
    - This results in using a long-term stable Q-value for (s',a')
- Why?
  - Decouples action choice from Q-value generation
  - Reduces overestimation of Q-values
  - Brings more stability to network

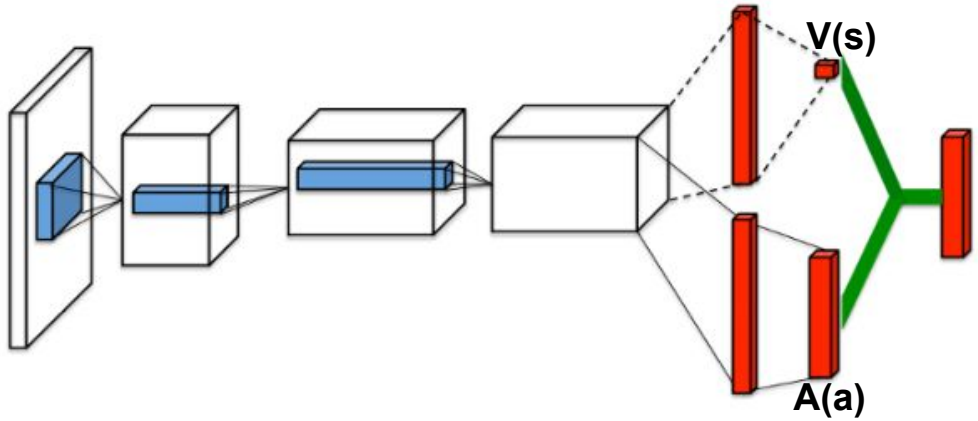
Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In AAAI, pages 2094–2100, 2016.



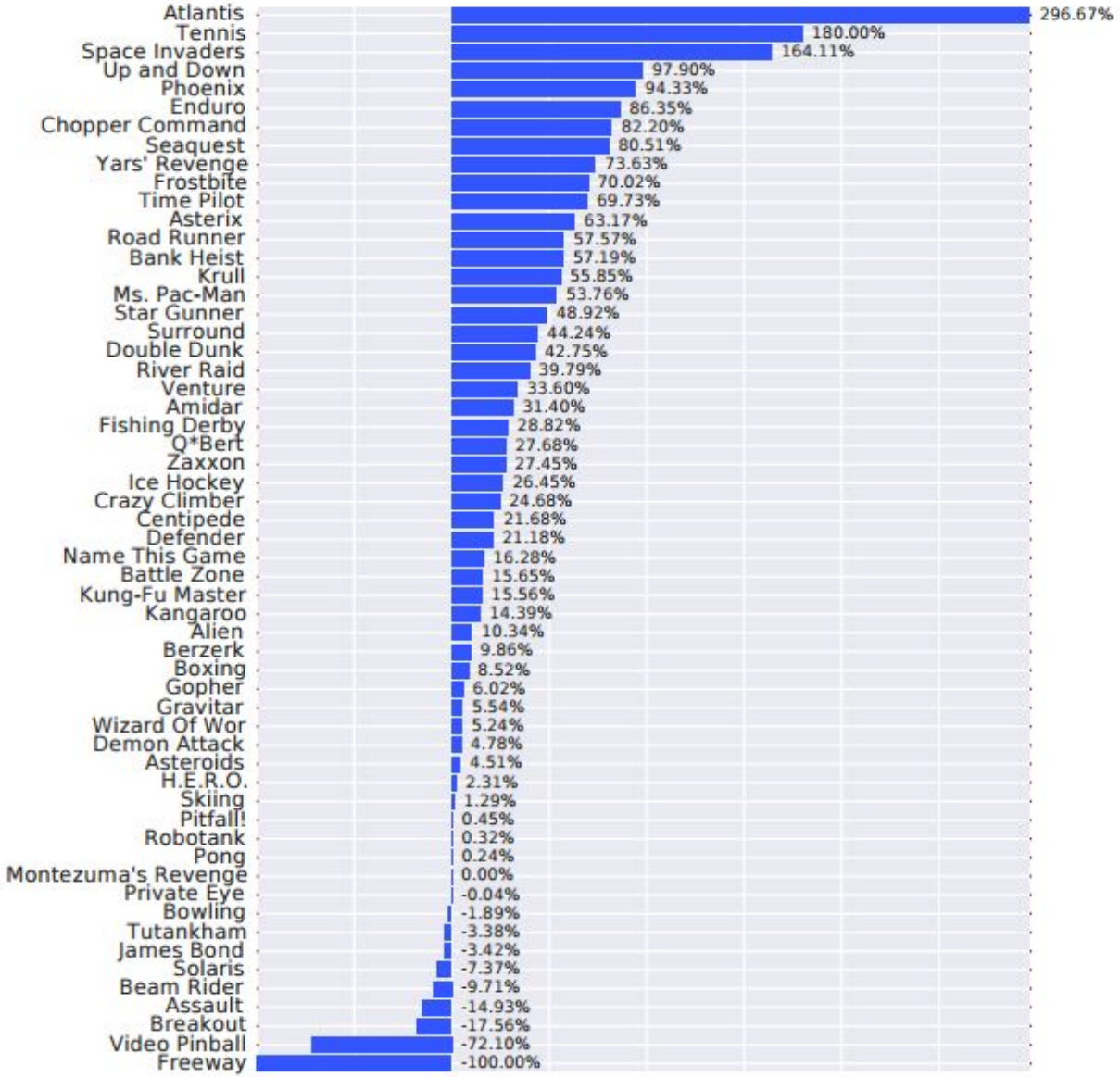
http://news.developer.nvidia.com/wp-content/uploads/2015/09/Google-Deep-Mind-Atari-chart1.png

Dueling Deep Q-Networks

- Uses methodologies and architecture from Double DQN
- Q-value corresponds to how beneficial it is to take action ***a*** in state ***s***
  - This can be split into two values: *V(s)*, *A(a)*
  - *V(s)* → how good it is to be in a given state
  - *A(a)* → advantage of taking action ***a*** compared to other possible actions
  - *Q(s,a) = V(s) + A(a)*



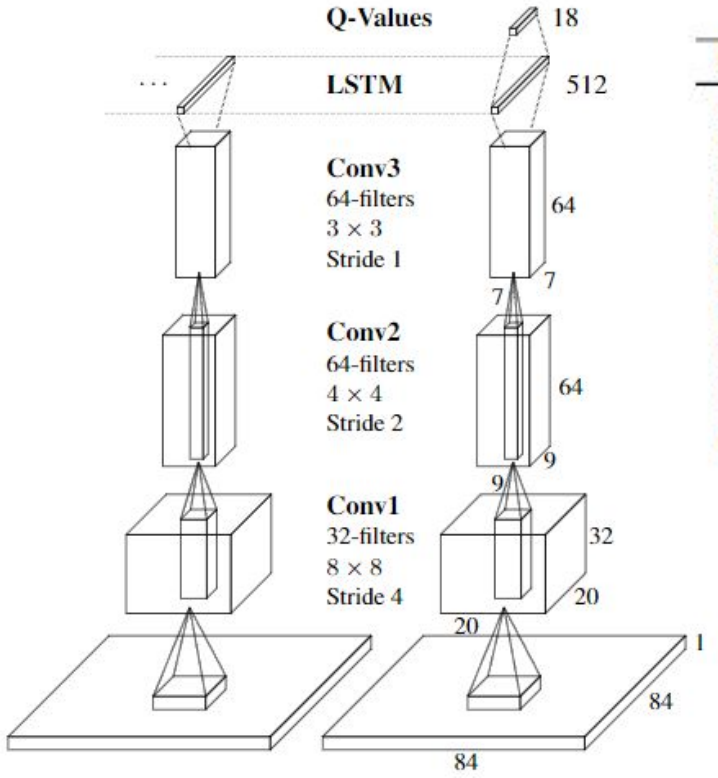
Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581, 2015.



https://lh5.googleusercontent.com/E2xaEVNSVU23vJuu3d1CvxMWvzFT0vOy1X_VD5eXFQlQz_81I3P-1aYovI7OVI7bzJIutZoHZrrP2xfJsqlDWzq7kILQMmzhsEF0aI9Iat6q1CN5x4r3eYypU09jive0RGcfUg4I

Deep Recurrent Q-Networks (DRQN)

- DQNs fail in Partially Observable MDP (POMDP)
- Need to keep track of sequences and past state decisions
- Add a Long Short-Term Memory (LSTM) layer to Dueling DQN



Game	DRQN ±std	DQN ±std Mnih et al.
Asteroids	1020 (±312)	1629 (±542)
Beam Rider	3269 (±1167)	6846 (±1619)
Bowling	62 (±5.9)	42 (±88)
Centipede	3534 (±1601)	8309 (±5237)
Chopper Cmd	2070 (±875)	6687 (±2916)
Double Dunk	**-2** (±7.8)	-18.1 (±2.6)
Frostbite	**2875** (±535)	328.3 (±250.5)
Ice Hockey	-4.4 (±1.6)	-1.6 (±2.5)
Ms. Pacman	2048 (±653)	2311 (±525)

Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. CoRR, abs/1507.06527, 2015.

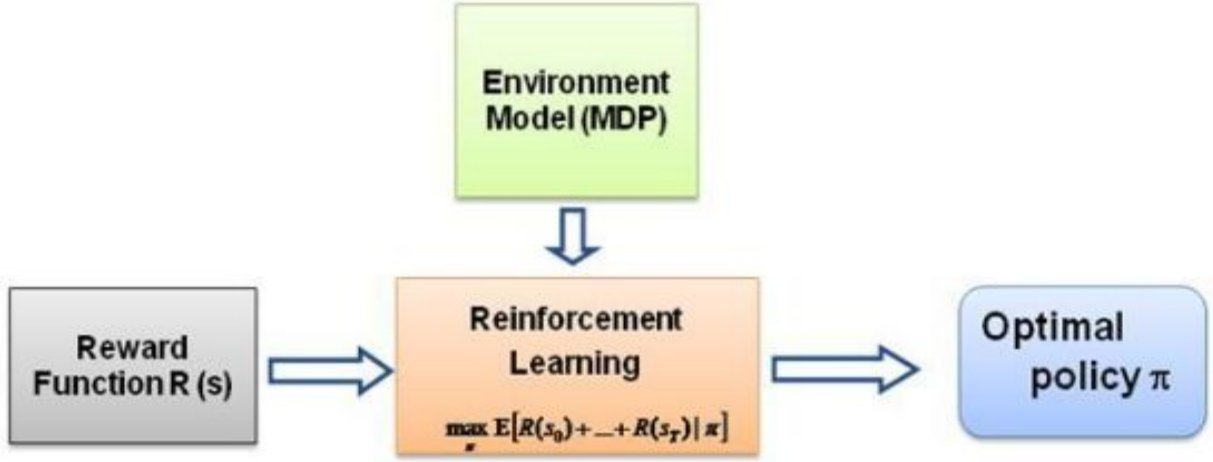
Agenda

- Reinforcement Learning
- **Apprenticeship Learning**
- Problem Statement
- Novel Contributions
- Algorithms and Methodologies
- Test Environment
- Results and Conclusion

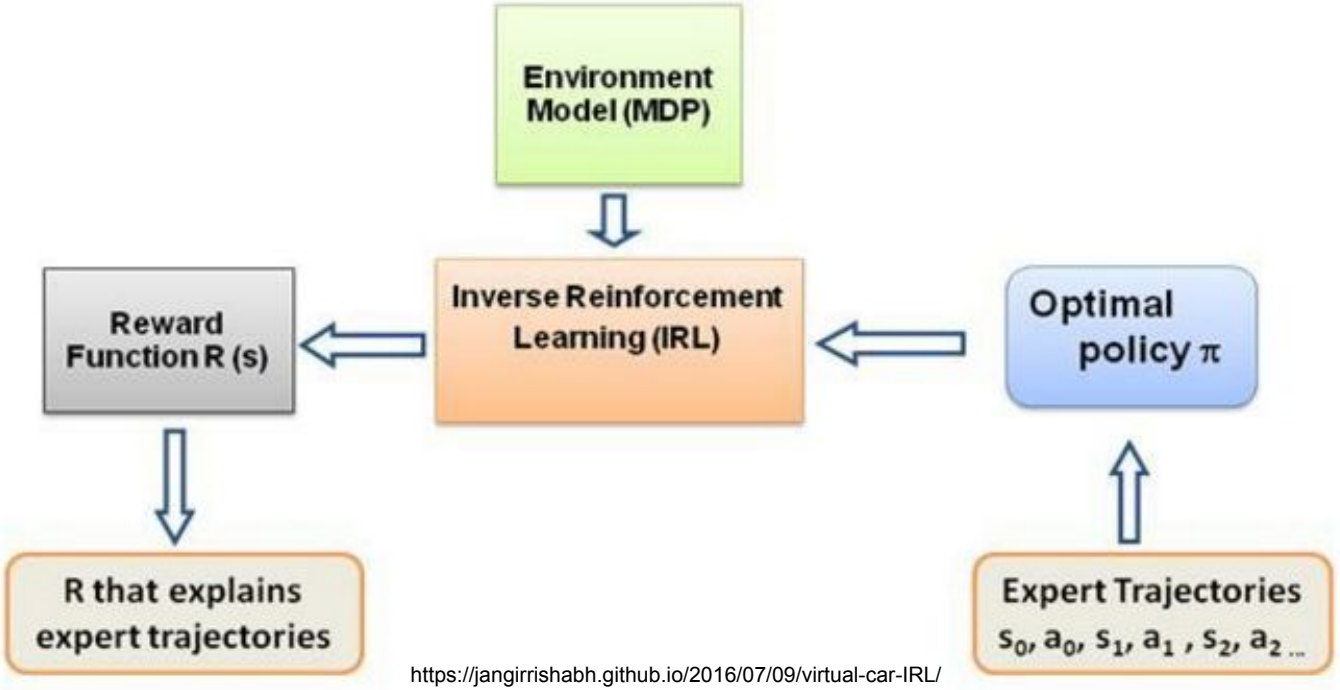
What is Apprenticeship (subset of Inverse Reinforcement) Learning?

- What if you do not know the specific reward/cost function of your system?
- Complex systems such as driving, walking, etc. may not have a simple cost function to minimize
- Create a system to learn an optimal policy or reward function from an expert to do a certain task

Reinforcement Learning



Apprenticeship Learning



https://jangirrishabh.github.io/2016/07/09/virtual-car-IRL/

Apprenticeship Learning

- First need to extract a policy from an expert set of demonstrations
- Then, learn a reward structure from the extracted policy
- This has been done using expert trajectories
  - Feeding a system the trajectories an expert takes
  - Using state visitation frequency to find a policy
  - Does not scale to diverse environments
  - Need to know state transition probabilities
- Not many raw pixel solutions

Related Works

Deep Apprenticeship Learning (DAL)

- This is an architecture to learn various games using raw pixels
  - Atari: Freeway, Space Invaders, Seaquest
- Trained in two steps:
  - Step 1: Deep Apprenticeship Q-Network (DAQN)
    - Extract policy from expert
  - Step 2: Deep Apprenticeship Reward Network (DARN)
    - Learn reward structure

Dejan Markovikj. Deep apprenticeship learning for playing games.
Master's thesis, Department of Computer Science, University of Oxford, 2014.

DAL - Step 1: DAQN

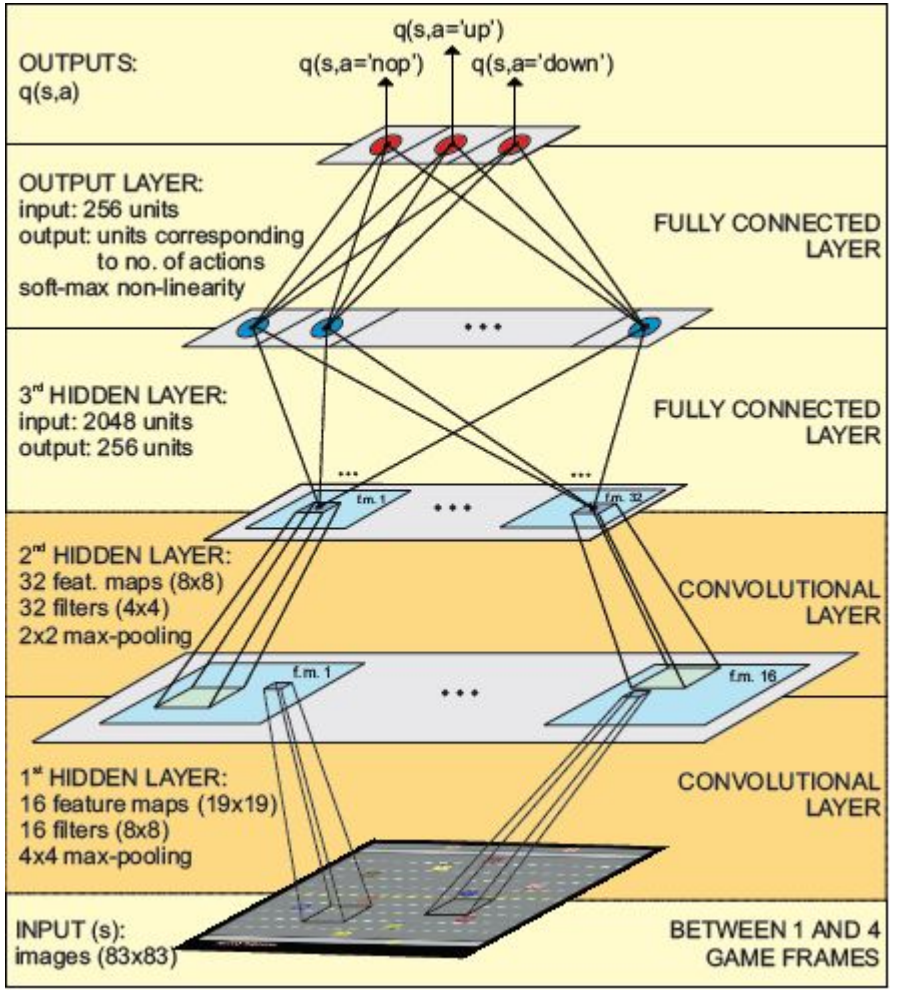
- Deep Apprenticeship Q-Network (DAQN)
- Takes in raw pixel inputs from expert demonstrations
- Learns the expert policy directly without a separate reward and value function
- Uses supervised learning for loss function

$$J = \Sigma(q(s,a) - \hat{q}(s,a))^2$$

where,

q = softmax array of predicted action

$\hat{q}$ = one-hot array of actual action taken



Dejan Markovikj. Deep apprenticeship learning for playing games. Master's thesis, Department of Computer Science, University of Oxford, 2014.

DAL - Step 2: DARN

- Deep Apprenticeship Reward Network (DARN)
- Takes in random state transitions ($s_1 \rightarrow s_2$)
- Learns the reward function
- Same architecture as DAQN (no transfer learning)

$$J = \| DARN(s,a) - [(DAQN^{PS}(s,a) - \gamma * max_{a'}\{DAQN^{PS}(s',a')\}] \|_2$$

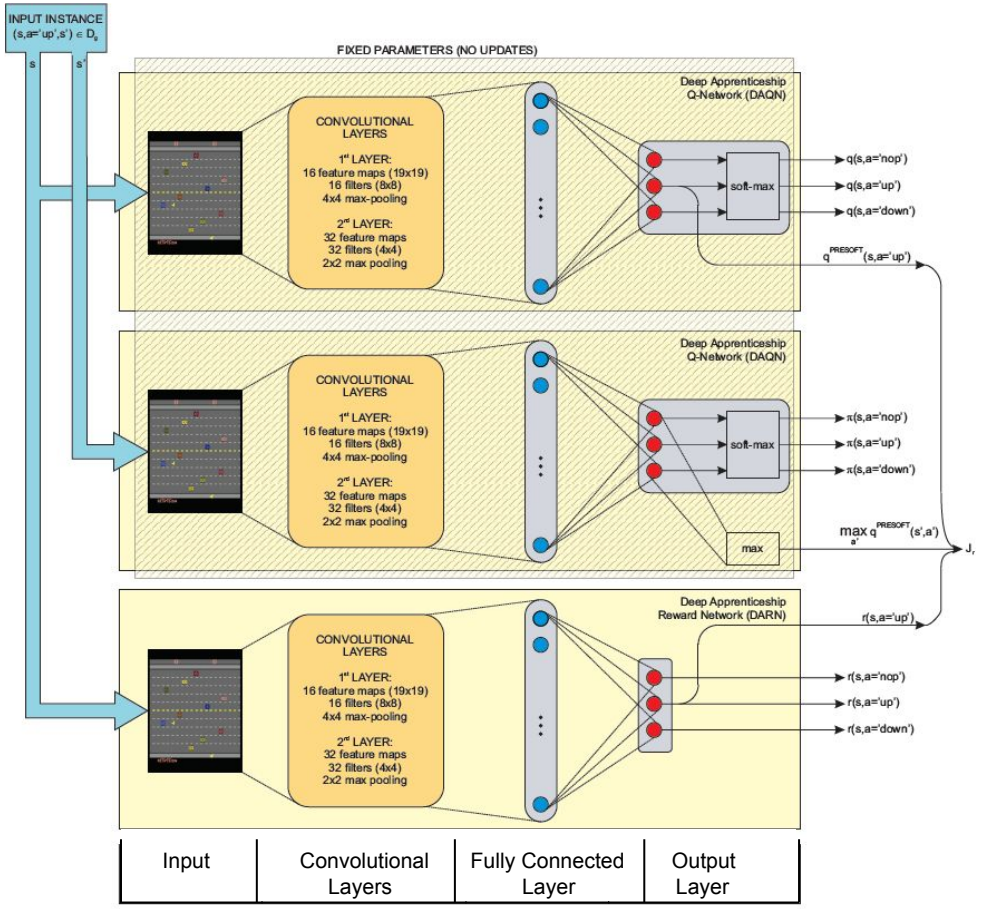
where,

$DARN(s,a) \rightarrow$ output of DARN for state s at action a

$DAQN^{PS} \rightarrow$ pre-softmax output of DAQN for state s and action a

$s = s_1$, $s' = s_2$

- When compared to Bellman Equation: $J = (Q_{target}(s,a) - Q(s,a))$
  - $DARN(s,a) \rightarrow Q_{target}(s,a)$, what you want your Q-value for (s,a) to be
  - $DAQN^{PS} - \gamma * max_{a'}\{DAQN^{PS}(s',a')\} \rightarrow Q(s,a)$, the current Q-value for (s,a)



Dejan Markovikj. Deep apprenticeship learning for playing games. Master's thesis, Department of Computer Science, University of Oxford, 2014.

Atari Freeway Results

Method	Reported score
random agent	0
human player	32
Sarsa	11
DAL	17

Dejan Markovikj. Deep apprenticeship learning for playing games.
Master's thesis, Department of Computer Science, University of Oxford, 2014.

https://atariage.com/2600/screenshots/s_Freeway_3.png

Deep Q-Learning from Demonstrations (DQfD)

- Utilizing DQNs with transfer learning
- First learns from expert demonstrations
- Then learns from experience
- Uses shared experience replay
  - Expert experiences (10%)
  - Agent experiences (90%)
- Still need to know reward structure
- Tested on Atari games

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, et al. Learning from demonstrations for real world reinforcement learning. arXiv preprint arXiv:1704.03732, 2017.

Game	DQfD	Double DQN	Imitation
Alien	**577.1**	280.1	473.9
Amidar	**250.4**	76.3	175.0
Assault	1017.4	**1384.9**	634.4
Asterix	2353.3	**4715.4**	279.9
Asteroids	**2507.8**	914.6	1267.3
Atlantis	**17647.0**	13494.8	12736.6
Bank Heist	**106.2**	8.7	95.2
Battle Zone	12486.1	3456.4	**14402.4**
Beam Rider	464.3	**748.8**	365.9
Bowling	46.5	28.5	**92.6**
Boxing	**89.1**	85.2	7.5
Breakout	**95.8**	5.3	3.5
Chopper Command	**2989.3**	2582.3	2485.7
Crazy Climber	103980.9	**108450.5**	14051.0
Defender	**7607.6**	3505.7	3819.1
Demon Attack	186.0	**405.1**	147.5
Double Dunk	**-16.9**	-20.2	-21.4
Enduro	624.8	**736.8**	134.8
Fishing Derby	-18.0	**-13.5**	-74.4
Freeway	**30.9**	28.5	22.7
Gopher	**9079.5**	4909.8	1142.6
Gravitar	245.1	35.6	**248.0**
Hero	**20428.2**	5373.0	5903.3
Ice Hockey	-9.8	**-4.7**	-13.5
James Bond	145.5	6.5	**262.1**
Kangaroo	1311.2	**1779.2**	917.3
Krull	1054.8	1880.2	**2216.6**
Kung Fu Master	**12328.6**	6677.8	556.7
Montezuma's Revenge	**780.9**	0.0	576.3
Ms Pacman	680.6	308.8	**692.4**
Name This Game	**4376.5**	4171.5	3745.3
Pitfall	-124.6	-26.3	**182.8**
Pong	**15.2**	13.6	-20.4
Private Eye	38280.5	-111.3	**42749.6**
Q-bert	2211.6	245.9	**5133.8**
River Raid	2368.0	**3202.6**	2148.5
Road Runner	38041.5	**39988.2**	8794.9
Seaquest	181.2	**1113.9**	195.6
Solaris	3107.9	221.8	**3589.6**
Up N Down	**10265.1**	8522.9	1816.7
Video Pinball	**10926.2**	7135.5	10655.5
Yars' Revenge	4764.3	**5731.8**	4225.8

Agenda

- Reinforcement Learning
- Apprenticeship Learning
- **Problem Statement**
- Novel Contributions
- Algorithms and Methodologies
- Test Environment
- Results and Conclusion

Motivation

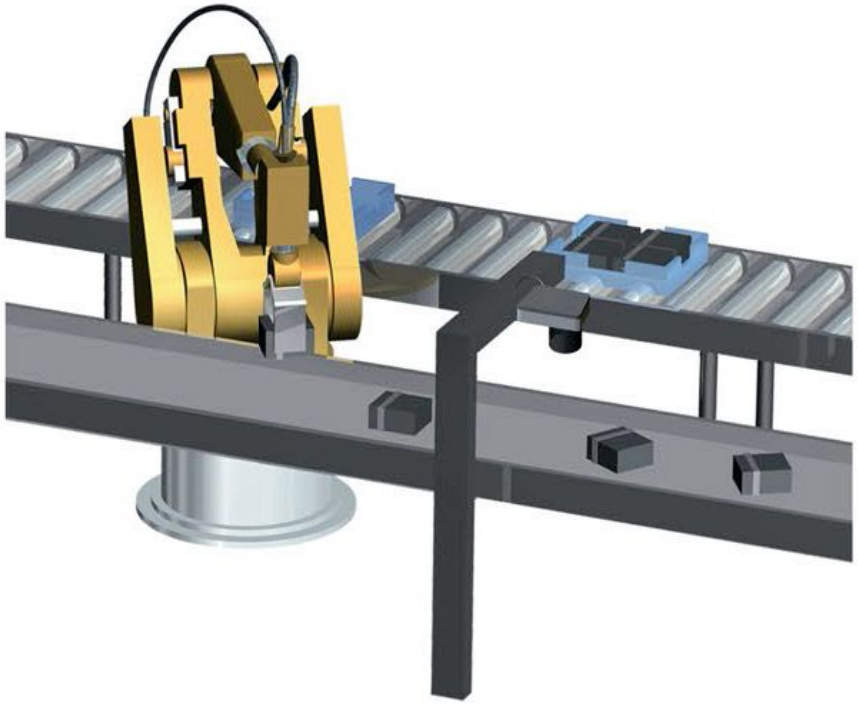
Teaching a system to perform a task given only videos of an expert doing the task well and optimally.

Examples:

1. Robotic surgical system learning to make proper cuts
2. Factory robot learning how to assemble a part
3. Autonomous vehicles learning to drive

Task Completion vs Task Understanding

- Task completion measures how well a system can complete a specific task.
  - Example: A robot picking up and placing a part
- Task understanding measures how well a system can generalize the task needing to be completed.
  - Example: A robot picking up, orienting, and placing a part in the correct box



http://www.adlinktech.com/industrial_automation/img/20130412-2.jpg?20150123

Agenda

- Reinforcement Learning
- Apprenticeship Learning
- Problem Statement
- **Novel Contributions**
- Algorithms and Methodologies
- Test Environment
- Results and Conclusion

Novel Contributions (1 of 2)

- **Reward Abstraction** → Task level abstraction of rewards (+ if complete, − if failed, 0 otherwise) for training on expert data and training the agent
- **Scheduled Shared Experience Replay** → Combining expert experiences and agent experiences with the amount of expert experiences used in replay decreasing over multiple epochs
- **Target Q-Network Implementation** → Implementing methods in DQN of stabilizing Q-networks with a target network updating frequency

Novel Contributions (2 of 2)

- **Dueling DQN Utilization** → Utilizing the concept of split value and advantage Q-values within expert data training and agent training
- **DRQN Comparison** → Utilizing the concept of recurrent neural networks and LSTMs to learn sequential tasks by adding an LSTM layer to the end of the Dueling DQN architecture, this will then be compared to the Dueling DQN architecture without the LSTM layer

Agenda

- Reinforcement Learning
- Apprenticeship Learning
- Problem Statement
- Novel Contributions
- **Algorithms and Methodologies**
- Test Environment
- Results and Conclusion

Proposed Architectures

- Deep Apprenticeship Learning (DAL)
  - DAL without any modifications
  - DAL without a pooling layer*
  - DAL with transfer learning from DAQN to DARN*
  - DAL with Bellman Equations and Reward Abstraction*
- Deep Q-Network Apprenticeship Learning
  - DQN-AL: Uses DQN architecture and shared experience replay*
    - DQN-AL with Scheduled Shared Experience Replay*
  - Dueling DQN-AL: Uses Dueling DQN architecture*
    - Dueling DQN-AL with Scheduled Shared Experience Replay*
  - DRQN-AL: Uses DRQN architecture*

* = indicates novel solutions

Deep Q-Network Apprenticeship Learning

$$loss = \lambda_1(Q_{target}(s,a) - Q(s,a))^2 + \lambda_2 cse(a^{predicted}, a_E)$$

where,

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

Q(s,a) is the output of the network for state s and action a

$a^{predicted}$ *is the action found by the network to have the highest value for a particular state*

a_E is the action the expert took

cse → cross softmax entropy error

λ_1, λ_2 → hyper-parameters to weight each argument ($\lambda_1 + \lambda_2 = 1$)

$\lambda_2 = 0$, during agent training

Training on expert data

for $e \in$ number of expert training episodes **do**
- Sample a random batch of expert data $<s_E, a_E, s'_E, r_E>$
- Feed-forward s_E to obtain current Q-value (Q) for a_E
- Feed-forward s'_E to obtain the Q-values (Q') for each action
- $Q'_{max} = max(Q')$
- $Q_{target} = r_E(s_E, a_E) + \gamma * Q'_{max}$
- $loss = \lambda_1(Q_{target} - Q)^2 + \lambda_2 cse(a^{predicted}, a_E)$
- Update network with batch *loss*

Initialize environment to get initial state s

Agent Training

for $e \in$ number of agent training episodes **do**
- **if** s is a terminal state **then**
  - Initialize new environment to get initial state s
- Feed-forward state s to get action a
- Play a to get s' and r
- Store $<s, a, s', r>$ in agent replay memory (replace old memories if full)
- $s \leftarrow s'$
- **if** replay buffer is full **then**
  - Sample a random batch of expert data $<s_E, a_E, s'_E, r_E>$
  - Sample a random batch of agent data $<s, a, s', r>$
  - Combine batches into $<s_T, a_T, s'_T, r_T>$
  - Feed-forward s_T to obtain current Q-value (Q) for a_T
  - Feed-forward s'_T to obtain the Q-values (Q') for each action
  - $Q'_{max} = max(Q')$
  - $Q_{target} = r_T(s_T, a_T) + \gamma * Q'_{max}$
  - $loss = \lambda_1(Q_{target} - Q)^2$
  - Update network with batch *loss*

Generate agent experiences

Batch train on expert and agent experiences

Agenda

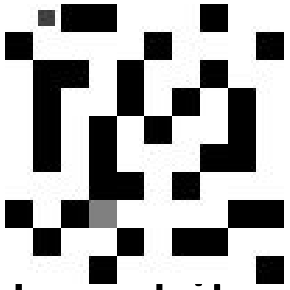
- Reinforcement Learning
- Apprenticeship Learning
- Problem Statement
- Novel Contributions
- Algorithms and Methodologies
- **Test Environment**
- Results and Conclusion

Challenge

- No datasets exist that have expert (demonstration) data and a simulation environment
- Had to create one from scratch
- Needed to be easily testable, easy to configure, easy to generate/simulate data

Maze World

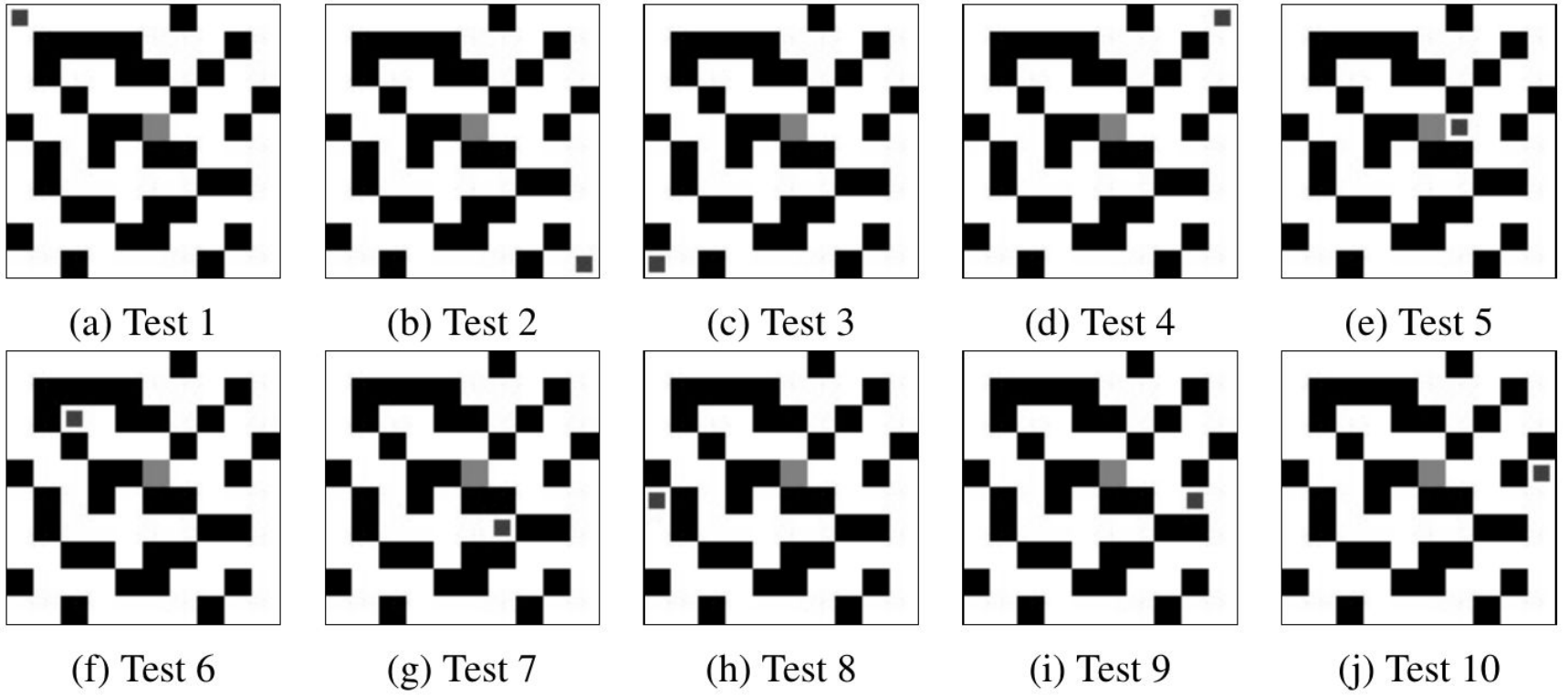
- 100 pixel by 100 pixel images of 10x10 mazes
- Expert data
  - Solved mazes using Dijkstra's algorithm



- Random data (state transitions for DAL architectures)
  - $s_1 \rightarrow s_2$

Testing Networks

- Task Completion

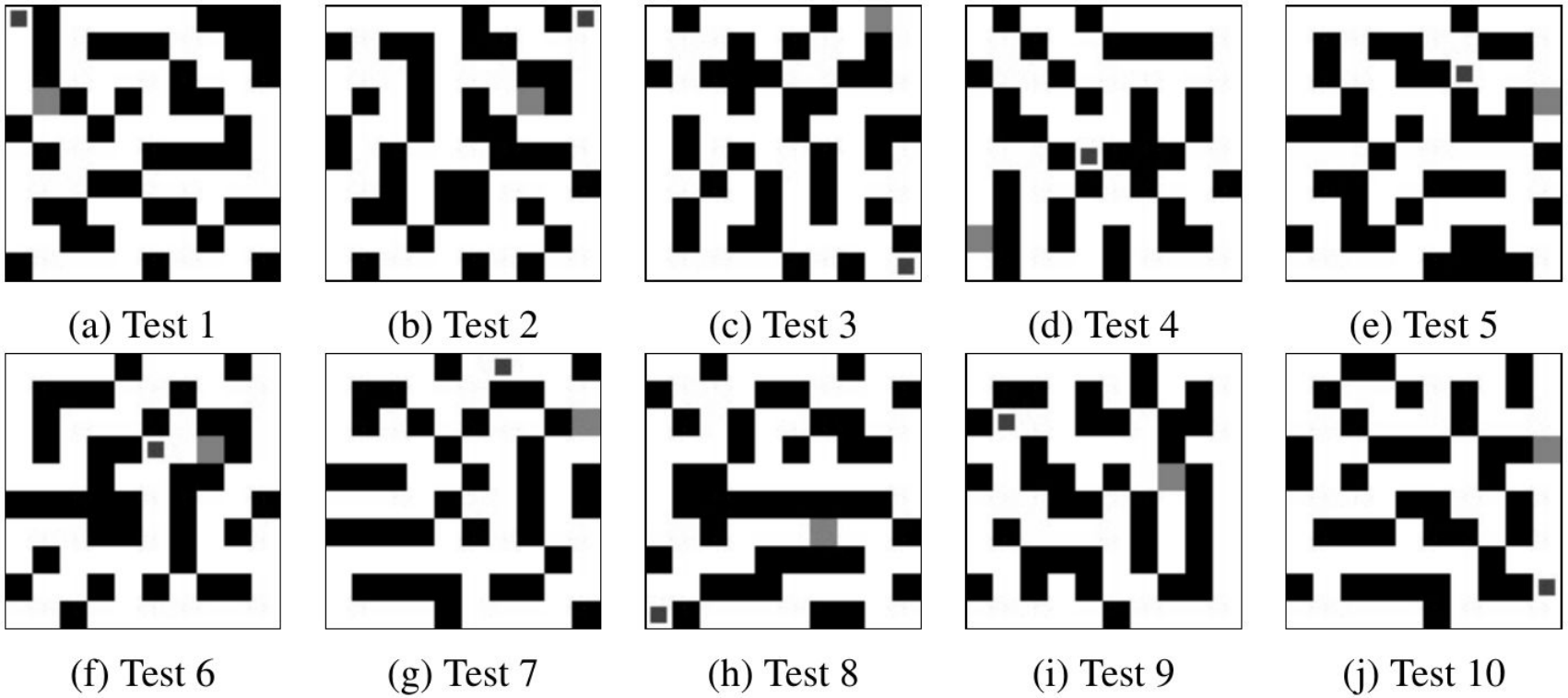


(a) Test 1 (b) Test 2 (c) Test 3 (d) Test 4 (e) Test 5

(f) Test 6 (g) Test 7 (h) Test 8 (i) Test 9 (j) Test 10

Testing Networks

- Task Understanding



(a) Test 1 (b) Test 2 (c) Test 3 (d) Test 4 (e) Test 5

(f) Test 6 (g) Test 7 (h) Test 8 (i) Test 9 (j) Test 10

Analysis

- **Completed Tests**: Number of mazes solved from the test set
- **Correct Action Prediction**: Percentage of correct actions predicted among the total number of correct actions

Agenda

- Reinforcement Learning
- Apprenticeship Learning
- Problem Statement
- Novel Contributions
- Algorithms and Methodologies
- Test Environment
- **Results and Conclusion**

Task Completion

Network	*Completed Tests*	*Correct Action Prediction*
DAL [18]	1/10	6.4% (8/125)
DAL [18] (no pooling)	0/10	5.6% (7/125)
DAL with Transfer Learning (no pooling)	2/10	10.4% (13/125)
DAL with Bellman Implementation	2/10	8.8% (11/125)
DQN-AL	4/10	26.4% (33/125)
DQN-AL with Scheduled Shared Experience Replay	6/10	36.0% (45/125)
Dueling DQN-AL	**10/10**	**100.0%** (125/125)
Dueling DQN-AL with Scheduled Shared Experience Replay	**10/10**	**100.0%** (125/125)
DRQN-AL	2/10	27.2% (34/125)

Novel Solutions

Task Completion Discussion - DAL

- Removing the pooling layer from the DAL did not result in better performance
  - The architecture is too simple to characterize the input images
- Showed an increase in performance using transfer learning technique
  - Gave proof that the learning from expert step can help the agent to learn better, rather than keeping them as separate entities
- Adding the Bellman equations to the expert training showed a slight increase in performance, but the instability of Q-learning caused it to diverge quickly
- This network is too simple, thus cannot be expanded to complex task environments

Network	***Completed Tests***	***Correct Action Prediction***
DAL [18]	1/10	6.4% (8/125)
DAL [18] (no pooling)	0/10	5.6% (7/125)
DAL with Transfer Learning (no pooling)	2/10	10.4% (13/125)
DAL with Bellman Implementation	2/10	8.8% (11/125)

Task Completion Discussion - DQN

- DQN architecture showed that it can learn from raw pixel inputs
  - This was proven using the Atari dataset, this network was made for this type of application
- Scheduled Shared Experience proved to be an enhancement
  - Allowed the DQN-AL to have better performance by acting as a guide throughout the agent's learning.
- Dueling DQN-AL showed the best performance due to its separation of Value and Advantage
  - This shows the robustness of this network modification, the agent learns the value of its current state regardless of the action it will take.
- DRQN-AL performed the worst
  - This type of network does not fare well in non-POMDP environments, it also is too complex for this type of application. DRQNs do not follow the rules of an MDP.

Network	***Completed Tests***	***Correct Action Prediction***
DQN-AL	4/10	26.4% (33/125)
DQN-AL with Scheduled Shared Experience Replay	6/10	36.0% (45/125)
Dueling DQN-AL	**10/10**	**100.0%** (125/125)
Dueling DQN-AL with Scheduled Shared Experience Replay	**10/10**	**100.0%** (125/125)
DRQN-AL	2/10	27.2% (34/125)

Task Understanding

Network	*Completed Tests*	*Correct Action Prediction*
DAL [18]	1/10	8.4% (8/95)
DAL [18] (no pooling)	1/10	14.7% (14/95)
DAL with Transfer Learning (no pooling)	2/10	11.6% (11/95)
DAL with Bellman Implementation	2/10	13.7% (13/95)
DQN-AL	1/10	11.6% (11/95)
DQN-AL with Scheduled Shared Experience Replay	1/10	15.8% (15/95)
Dueling DQN-AL	**3/10**	**23.2%** (22/95)
Dueling DQN-AL with Scheduled Shared Experience Replay	3/10	21.1% (20/95)
DRQN-AL	1/10	13.7% (13/95)

Novel Solutions

Task Understanding Discussion - DAL

- DAL showed poor performance
  - Each network modification performed relatively the same, and beat the original DAL network
  - This type of network is not useful for task understanding
    - It cannot understand or learn the structure of the input images causing it to perform no better than randomly guessing

Network	***Completed Tests***	***Correct Action Prediction***
DAL [18]	1/10	8.4% (8/95)
DAL [18] (no pooling)	1/10	14.7% (14/95)
DAL with Transfer Learning (no pooling)	2/10	11.6% (11/95)
DAL with Bellman Implementation	2/10	13.7% (13/95)

Task Understanding Discussion - DQN

- DQN architecture proved to be an enhancement over the DAL network
- DQN-AL performed similarly to the DAL, this was most likely due to the fluctuating Q-values since this implementation did not use the Target Q-Network
- Scheduled Shared Experience proved to be an enhancement for the DQN-AL, but showed to have similar results for the Dueling DQN-AL architecture
- Dueling DQN-AL showed the best performance due to its separation of Value and Advantage
- DRQN-AL performed the worst as was expected due to its complexity and failure to adhere to the MDP rule

Network	*Completed Tests*	*Correct Action Prediction*
DQN-AL	1/10	11.6% (11/95)
DQN-AL with Scheduled Shared Experience Replay	1/10	15.8% (15/95)
Dueling DQN-AL	**3/10**	**23.2%** (22/95)
Dueling DQN-AL with Scheduled Shared Experience Replay	3/10	21.1% (20/95)
DRQN-AL	1/10	13.7% (13/95)

Conclusion

- **Reward abstraction** brought the ability to utilize DQNs and Bellman equations in the realm of apprenticeship learning

- **Scheduled Shared Experience Replay** helps to guide the learning of an agent, and models our thought process as humans
- **Dueling DQN architecture** proves to be the most robust, and is used as the standard for many reinforcement learning tasks
- **DRQN architecture** proves to be too complex of an architecture for this problem space

Future Work

- Generating human created datasets (timely)
- Introducing stochasticity to the expert dataset or in agent learning
- Expanding testing to a physical environment


Machine Intelligence Lab (MIL)
Rochester Institute of Technology


R·I·T

Questions

Hyper-Parameters

Deep Q-Learning from Demonstrations (DQfD)

- Loss function:
  - $J_{DQ}(Q) = (r(s,a) + \gamma * max_{a'}\{Q(s',a')\} - Q(s,a))^2$
    - $r(s,a) \rightarrow$ score from environment
  - $J_E(Q) = max_a\{Q(s,a) + l(s,a_E,a)\} - Q(s,a_E)$
    - $l(s,a_E,a) \rightarrow$ expert comparison,
      1 *if* $a_E = a$,
      0 otherwise
  - $J(Q) = J_{DQ}(Q) + \lambda_1 J_E(Q) + \lambda_2 J_{L2}(Q)$
    - $J_{L2}(Q) \rightarrow$ L2 regularization of Q network
    - $\lambda_1, \lambda_2 \rightarrow$ weights

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, et al. Learning from demonstrations for real world reinforcement learning. arXiv preprint arXiv:1704.03732, 2017.