COL778: Principles of Autonomous Systems
Semester II, 2023-24

Deep Q-Learning

Rohan Paul

Outline

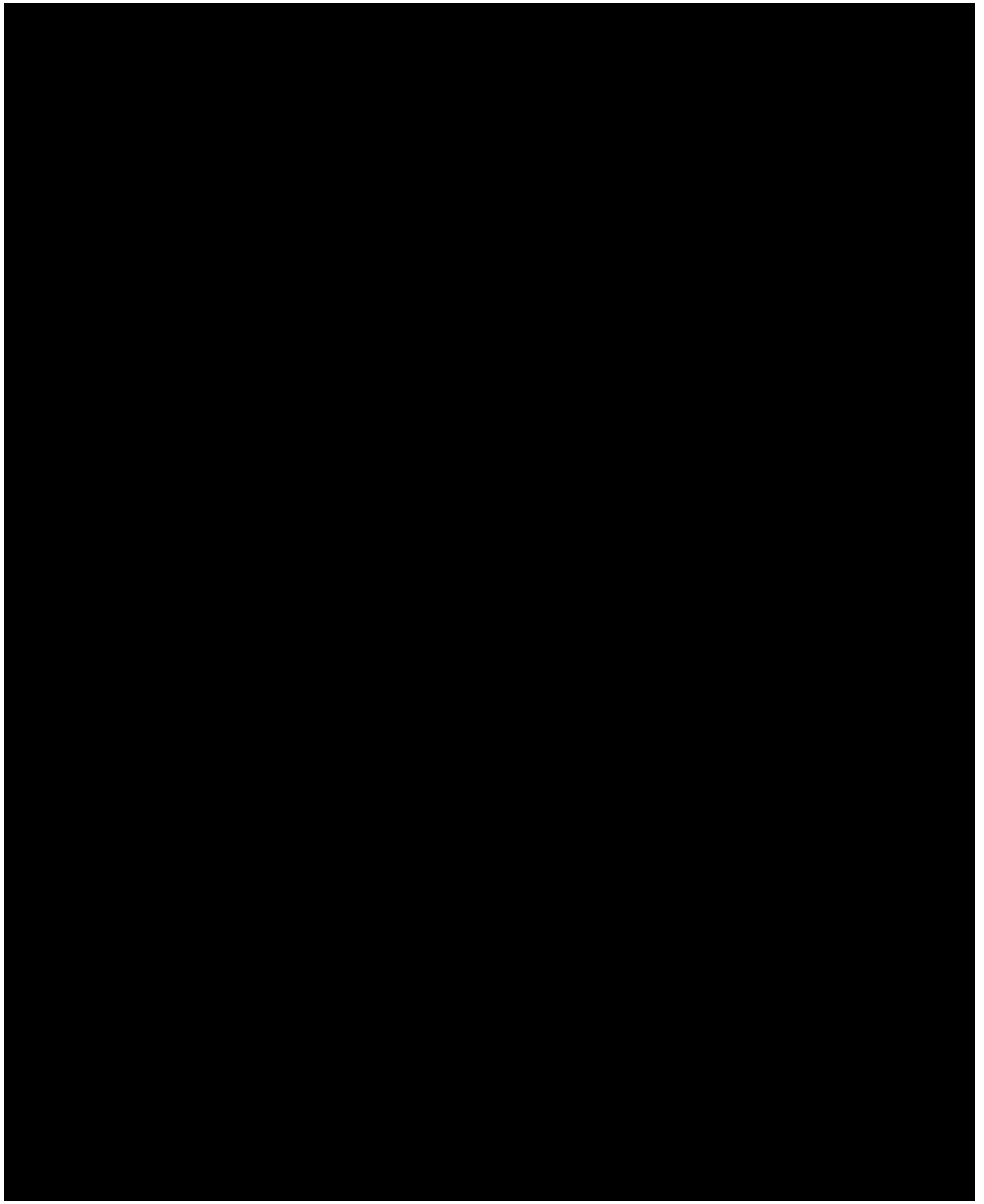
- Last Class
  - Reinforcement Learning - Introduction
- This Class
  - Deep Q-Learning
- Reference Material
  - Please follow the notes as the primary reference on this topic.

Acknowledgements

These slides are intended for teaching purposes only. Some material has been used/adapted from web sources and from slides by Nicholas Roy, Wolfram Burgard, Dieter Fox, Sebastian Thrun, Siddharth Srinivasa, Dan Klein, Pieter Abbeel, Max Likhachev, Alexander Amini, Jeanette Bohg, Dorsa Sadigh and Marco Pavone and others.

Example: Learning to play a game

- The game of break out requires learning ways to hit the ball so as to break a large number of tiles.
- Actions – left, right.
- One needs to learn how to hit to maximize the breaking.
- We learn via experience, here we want to use RL for the same.



https://www.youtube.com/watch?v=V1eYniJ0Rnk

Recap: Q-Values

- Q*(s,a) is the expected utility starting in state s, taking action a and (thereafter) acting optimally.

Bellman Equation:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*(s',a'))$$

Q-Value Iteration:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$$

Learning the Q-function

- In the RL setting we don’t have access to the reward model/transition model a-priori.
- How can the agent learn the Q-function by collecting experience?
- Once the Q-function is acquired the agent can decide the actions to take.
- What we need?
  - Estimating the Q-values
  - How to take actions (explore vs. exploit).

Learning the Q-function

- Core Idea: Learn Q(s, a) function from rollouts. Estimate the best that can be achieved by taking action a in state s.
- Derive the policy from the Q-function.

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

(state, action)

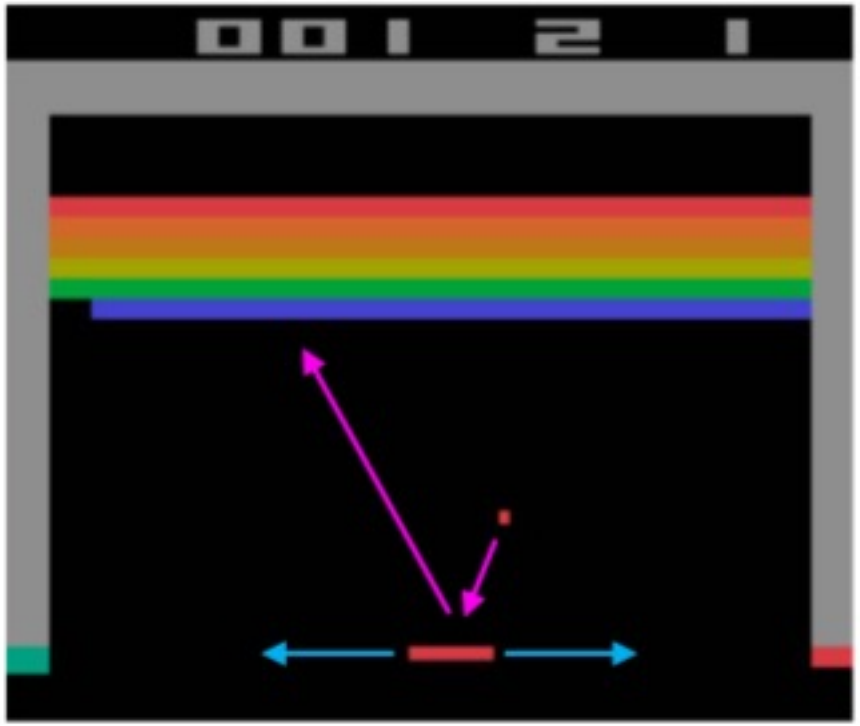
Ultimately, the agent needs a **policy $\pi(s)$,** to infer the **best action to take** at its state, s

Strategy: the policy should choose an action that maximizes future reward

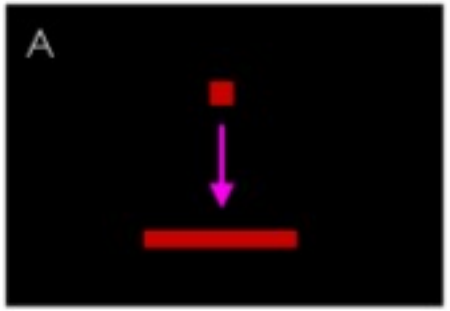
$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

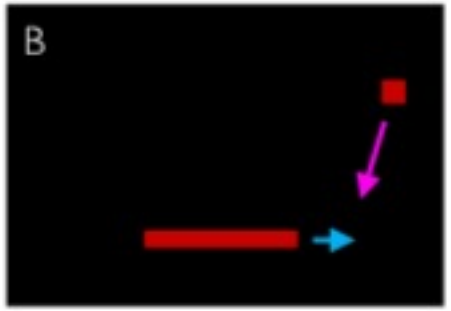
Learning Q-functions is non-trivial

Example: Atari Breakout



It can be very difficult for humans to accurately estimate Q-values

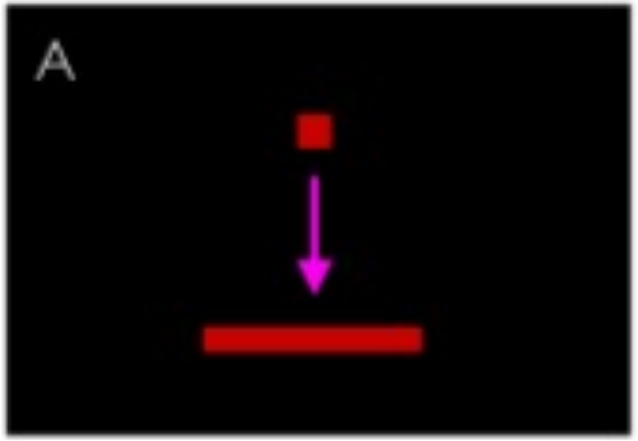




Which $(\mathbf{s}, \mathbf{a})$ pair has a higher Q-value?

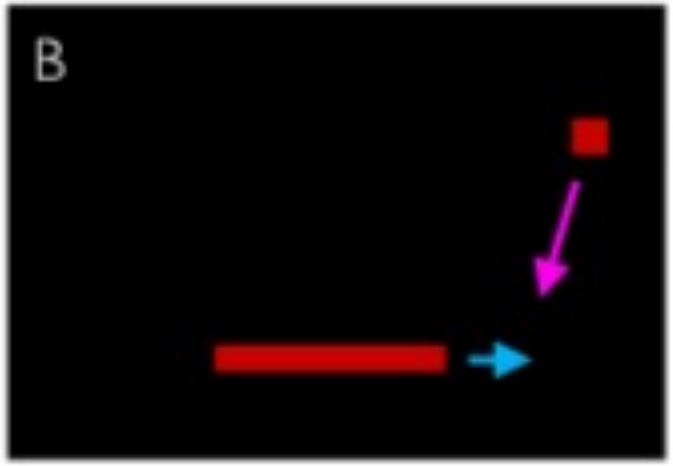
Q-value for an action

Example: Atari Breakout - Middle



Q-value for another action

Example: Atari Breakout - Side



(Tabular) Q-Learning

- Q-value iteration: $Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$
- Rewrite as expectation: $Q_{k+1} \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$
- (Tabular) Q-Learning: replace expectation by samples
  - For an state-action pair (s,a), receive: $s' \sim P(s'|s,a)$
  - Consider your old estimate: $Q_k(s,a)$
  - Consider your new sample estimate:

    $$\text{target}(s') = R(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$

  - Incorporate the new estimate into a running average:

    $$Q_{k+1}(s,a) \leftarrow (1-\alpha) Q_k(s,a) + \alpha \left[\text{target}(s') \right]$$

(Tabular) Q-Learning

Algorithm:

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s'

If s' is terminal:

$$\text{target} = R(s, a, s')$$

Sample new initial state s'

else:

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

$$Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha \, [\text{target}]$$

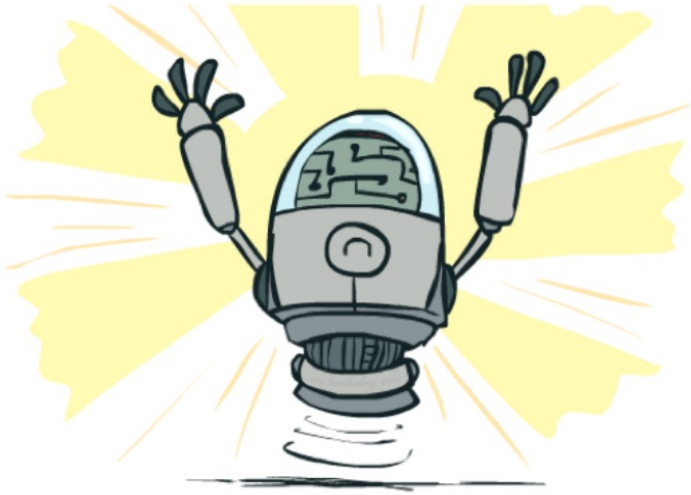
$$s \leftarrow s'$$

Deciding which actions to take?

- Choose random actions?
- Choose action that maximizes $Q_k(s, a)$ (i.e. greedily)?
- ε-Greedy: choose random action with prob. ε, otherwise choose action greedily

Q-learning properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
- This is called off-policy learning
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly



Issue of Generalization

- Generalization
  - Feature representation of states.
  - States that share features would share other properties as well.
  - Each state does not have to be dealt with independently.
  - Represent the Q or the V function as a linear combination of features.

Example: Atari Breakout - Side

Example: Atari Breakout - Middle

Approximate Q-Learning

- Instead of a table, we have a ***parametrized Q function***: $Q_\theta(s, a)$
  - Can be a linear function in features:

$$Q_\theta(s, a) = \theta_0 f_0(s, a) + \theta_1 f_1(s, a) + \cdots + \theta_n f_n(s, a)$$

  - Or a neural net, decision tree, etc.
- ***Learning rule***:
  - Remember: $\text{target}(s') = R(s, a, s') + \gamma \max_{a'} Q_{\theta_k}(s', a')$
  - Update:

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \left[\frac{1}{2} (Q_\theta(s, a) - \text{target}(s'))^2 \right] \Big|_{\theta=\theta_k}$$

Parameterizing Q function with a neural net

- Instead of a table, we have a parametrized Q function
  - E.g. a neural net $Q_\theta(s, a)$
- Learning rule:
  - Compute target:

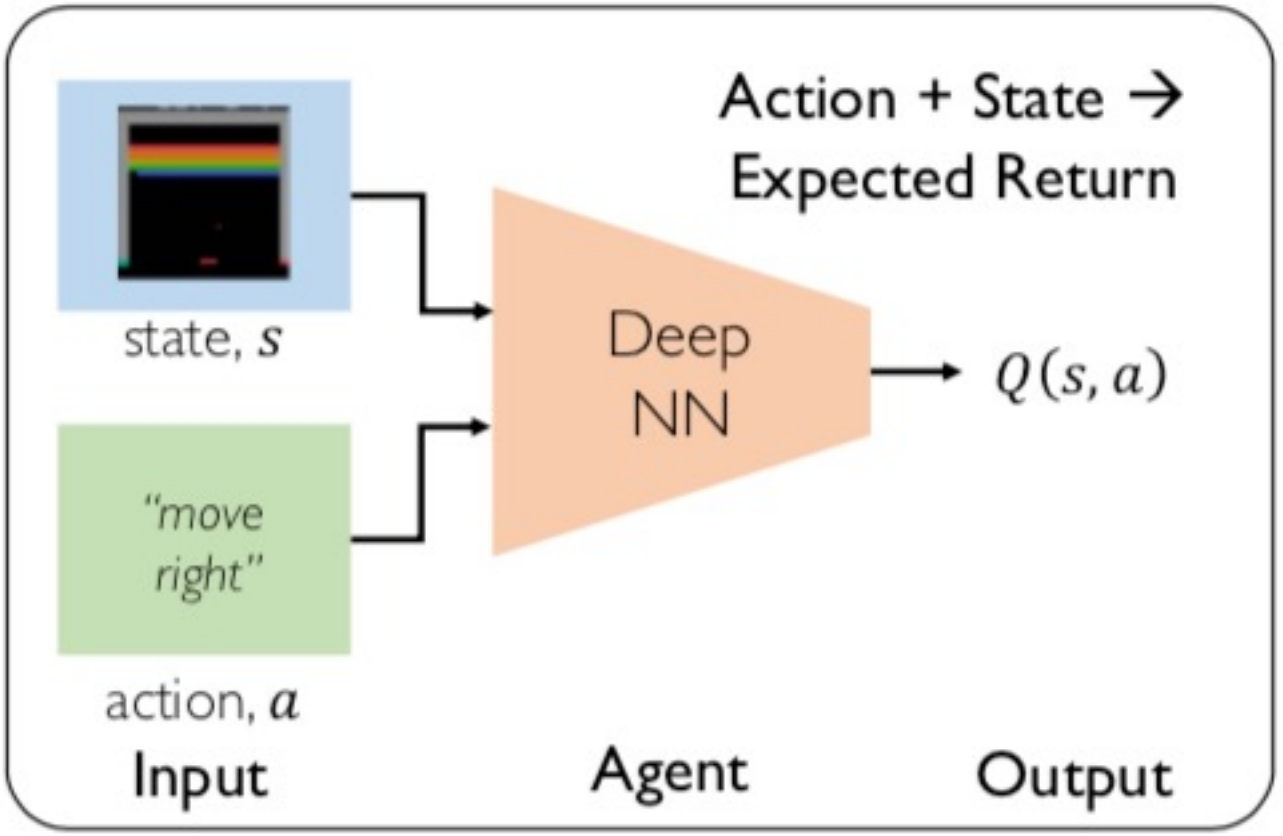
$$\text{target}(s') = R(s, a, s') + \gamma \max_{a'} Q_{\theta_k}(s', a')$$

  - Update Q-network:

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \left[\frac{1}{2} (Q_\theta(s, a) - \text{target}(s'))^2 \right] \Big|_{\theta=\theta_k}$$

Deep Q-Networks (DQNs)

Modeling the Q-function via a Neural Network.



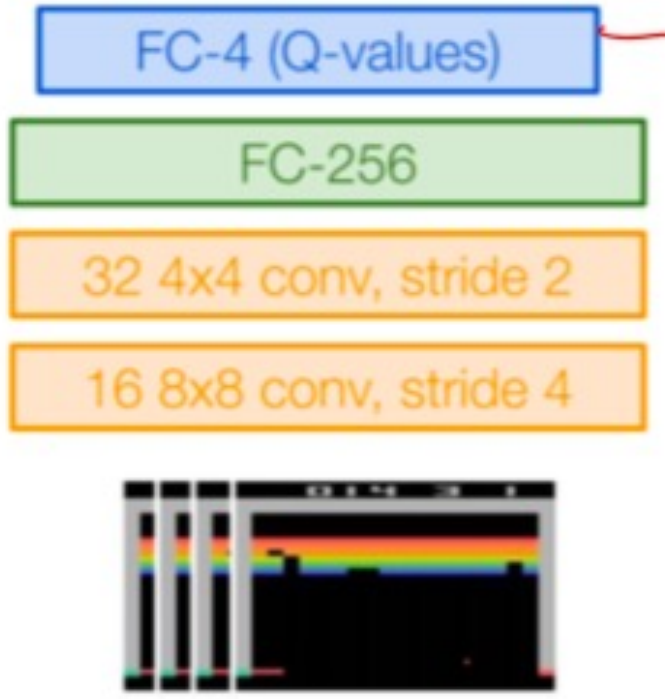
How to Encode an MDP as a NN input?

- Encoding the state-action pair
  - One network for each action a_j, that takes s as input and produces $Q(s,a_j)$ as output
  - One single network that takes s as input and produces a vector $Q(s, \cdot)$, consisting of the Q values for each action
  - One single network that takes s, a concatenated into a vector (if a is discrete, we would probably use a one-hot encoding, unless it
- How to decide which encoding to use?
  - First two network structures are only suitable for discrete (and not too big) action sets.
  - The last structure can be applied for continuous actions, but then it is difficult to find $\text{argmax}_A Q(s, a)$.
- Driven by experience
  - The more structure we encode easier to generalize

Deep Q-Networks (DQNs)

Output: Q(s, left), Q(s, right), …..

CNNs extensively used in computer vision
If we want to go from pixels to decisions, likely useful to leverage insights for visual input

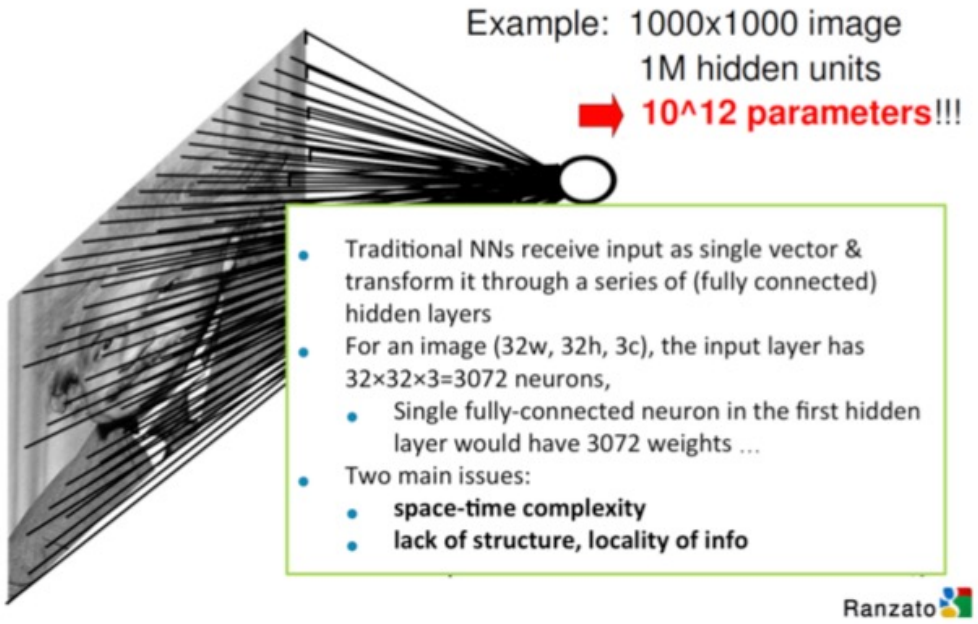


Current state s_t: 84*84*4 stack of last four frames. After RGB-> grayscale conversion, downsampling and cropping.

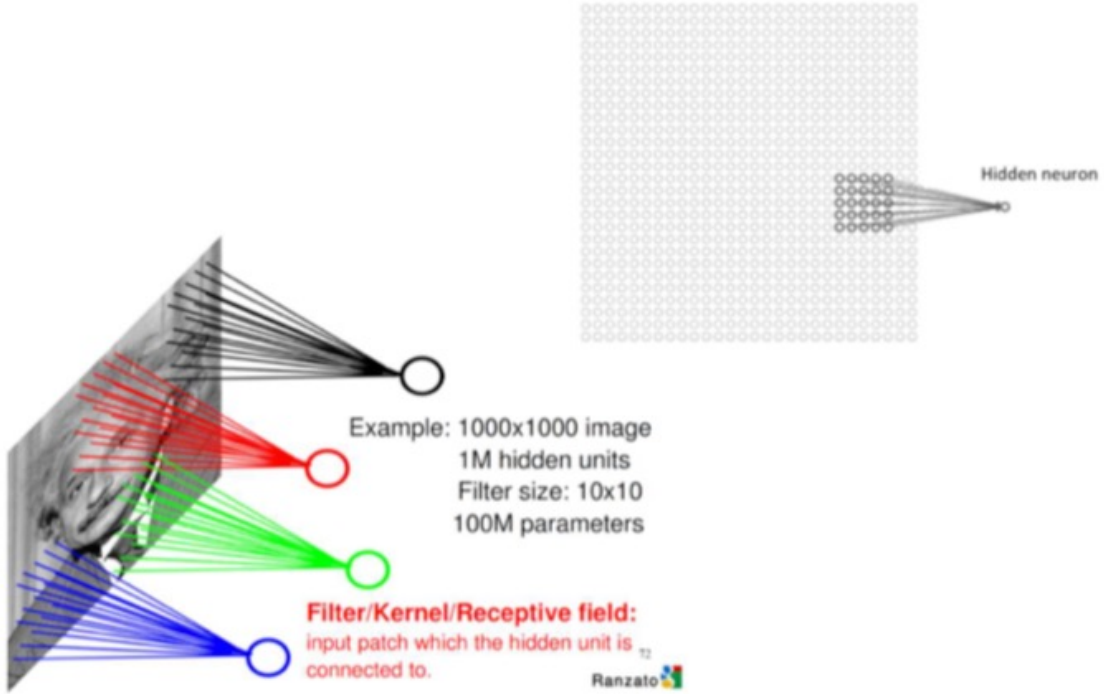
Minh, Nature 2015

Locality of Information: Receptive Fields

Fully connected network.

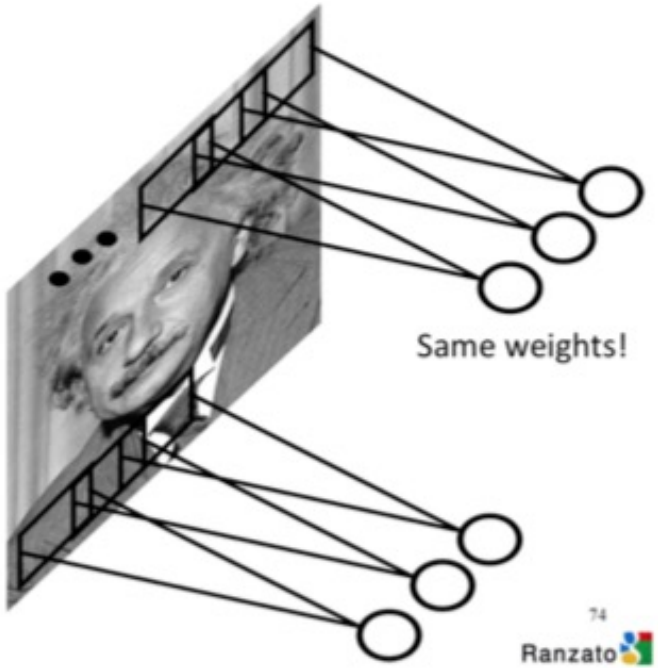


Convolutional NN

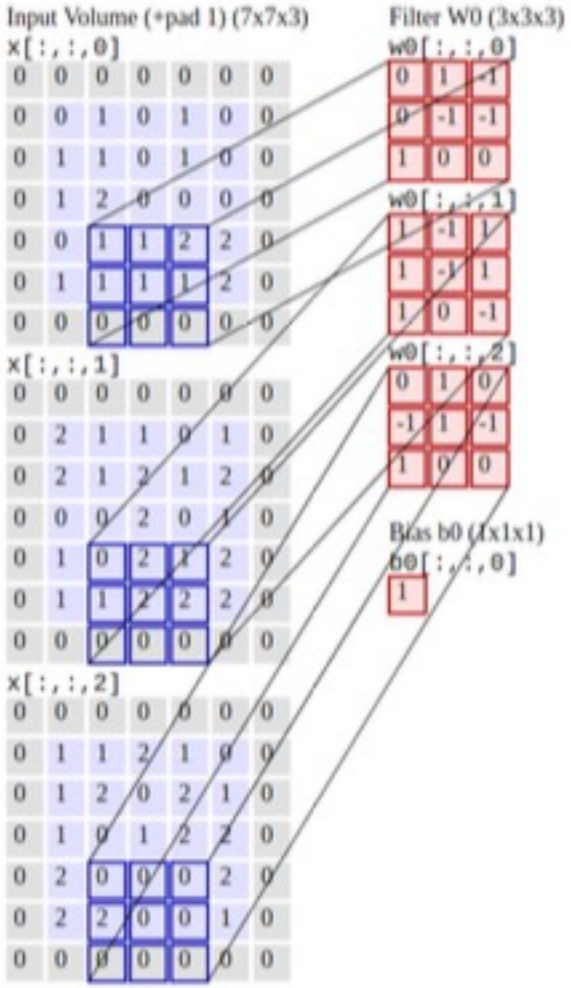


Convolutional NN

Feature Maps



- The map from the input layer to the hidden layer is therefore a feature map: all nodes detect the same feature in different parts
- The map is defined by the shared weights and bias
- The shared map is the result of the application of a convolutional filter (defined by weights and bias), also known as convolution with learned kernels



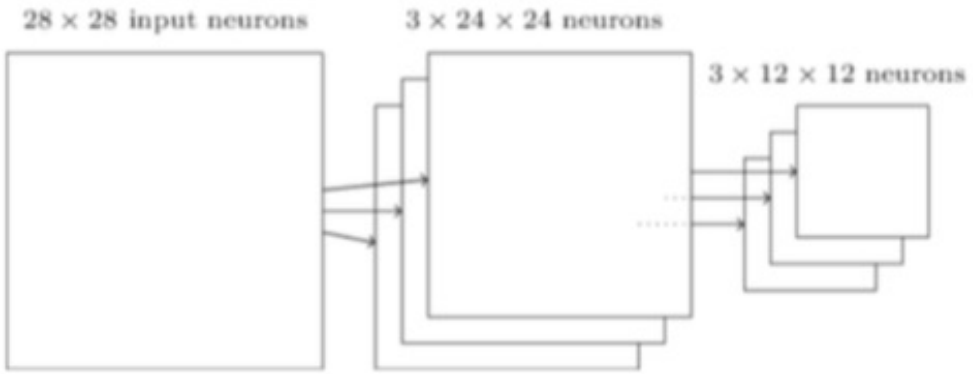
Convolutional NN

Pooling Layers

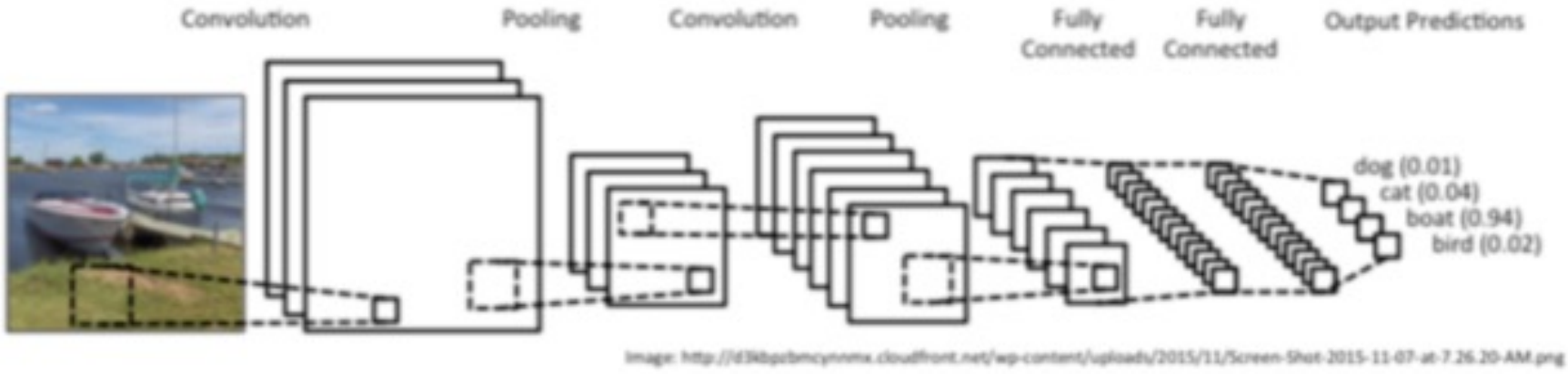
Pooling layers are usually used immediately after convolutional layers.

Pooling layers simplify / subsample / compress the information in the output from convolutional layer

A pooling layer takes each feature map output from the convolutional layer and prepares a condensed feature map

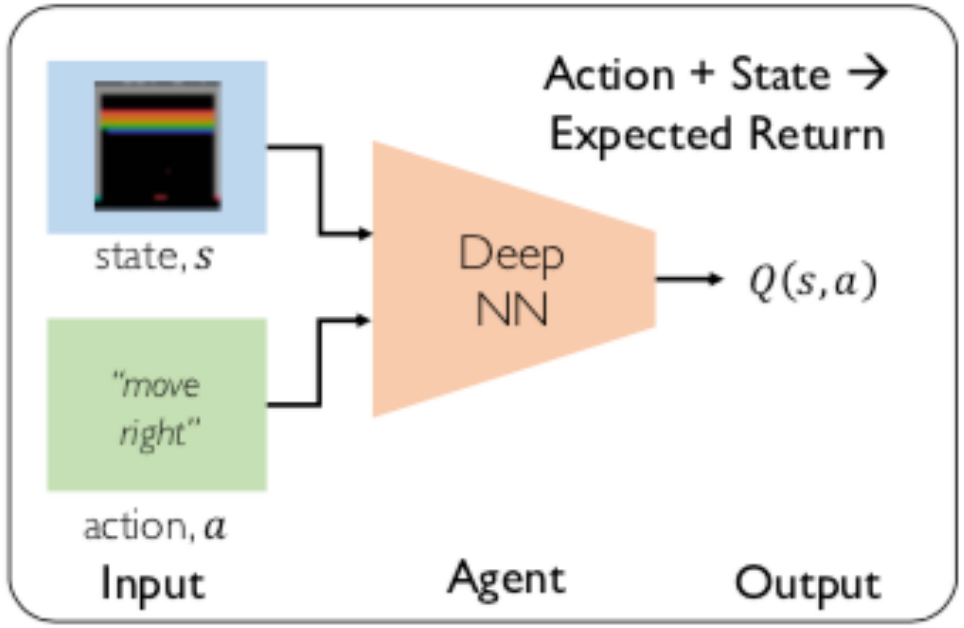


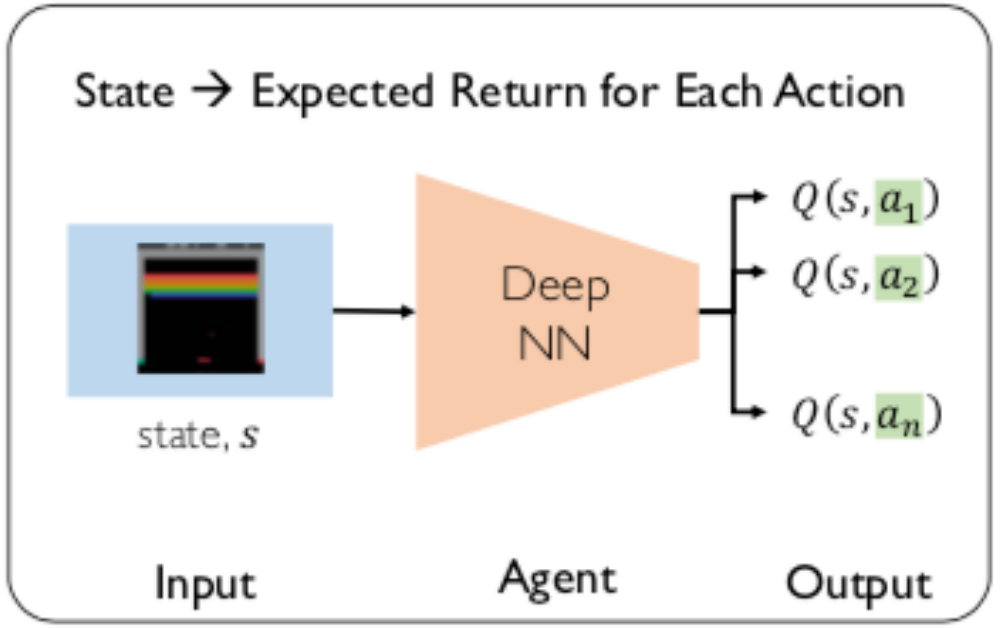
Convolution NN



- Consider local structure and common extraction of features
- Not fully connected. Locality of processing
- Weight sharing for parameter reduction
- Learn the parameters of multiple convolutional filter banks
- Compress to extract salient features & favor generalization

DQN Training: Target Value



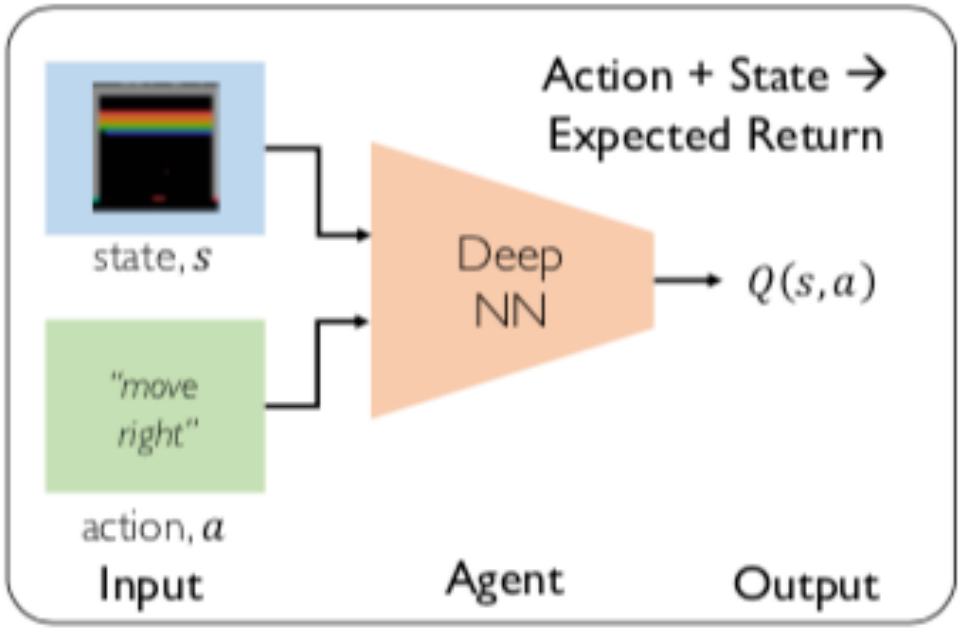


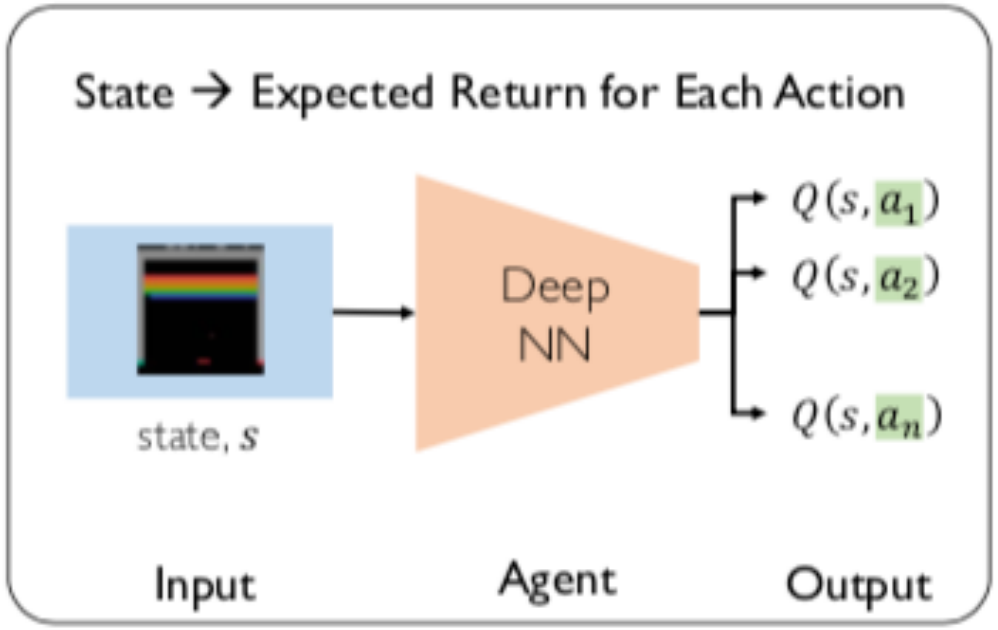
Training data: Experiences of the agent interacting with the environment. These are the values the agent is observing in the environment.

$$\overbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}^{\text{target}}$$

Take all the best actions → target return

DQN Training: Network Predictions



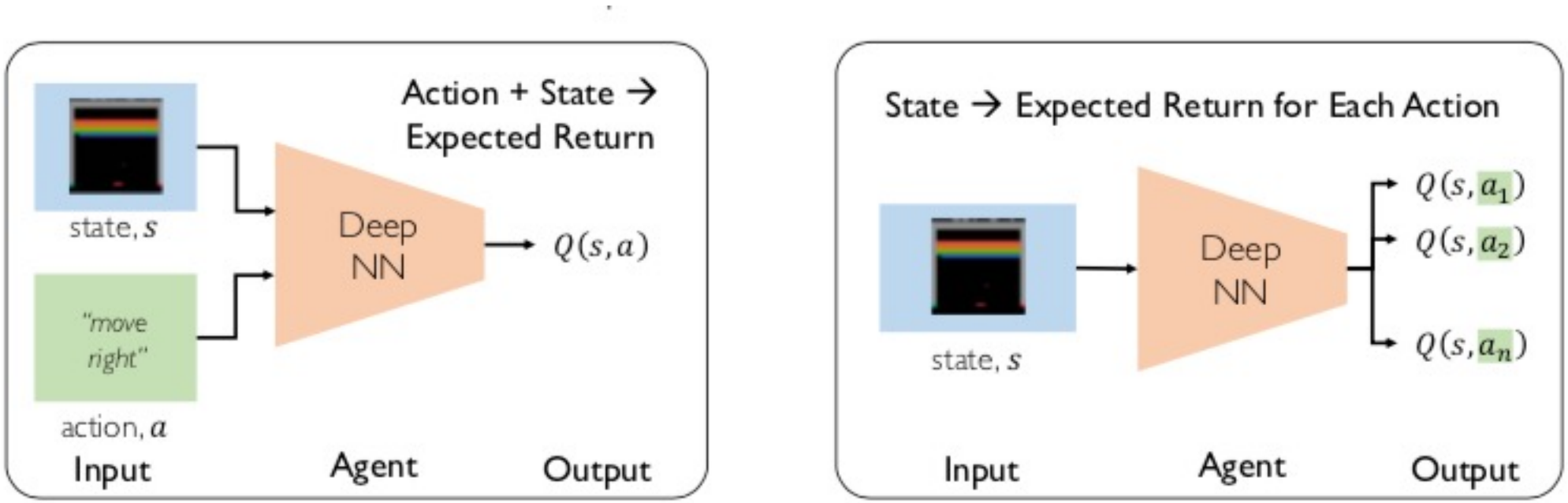


Prediction of the network is a Q (s, a) value.

$$\underbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}_{\text{target}} \quad \underbrace{Q(s,a)}_{\text{predicted}}$$

Network prediction

DQN Training: MSE Loss Minimization

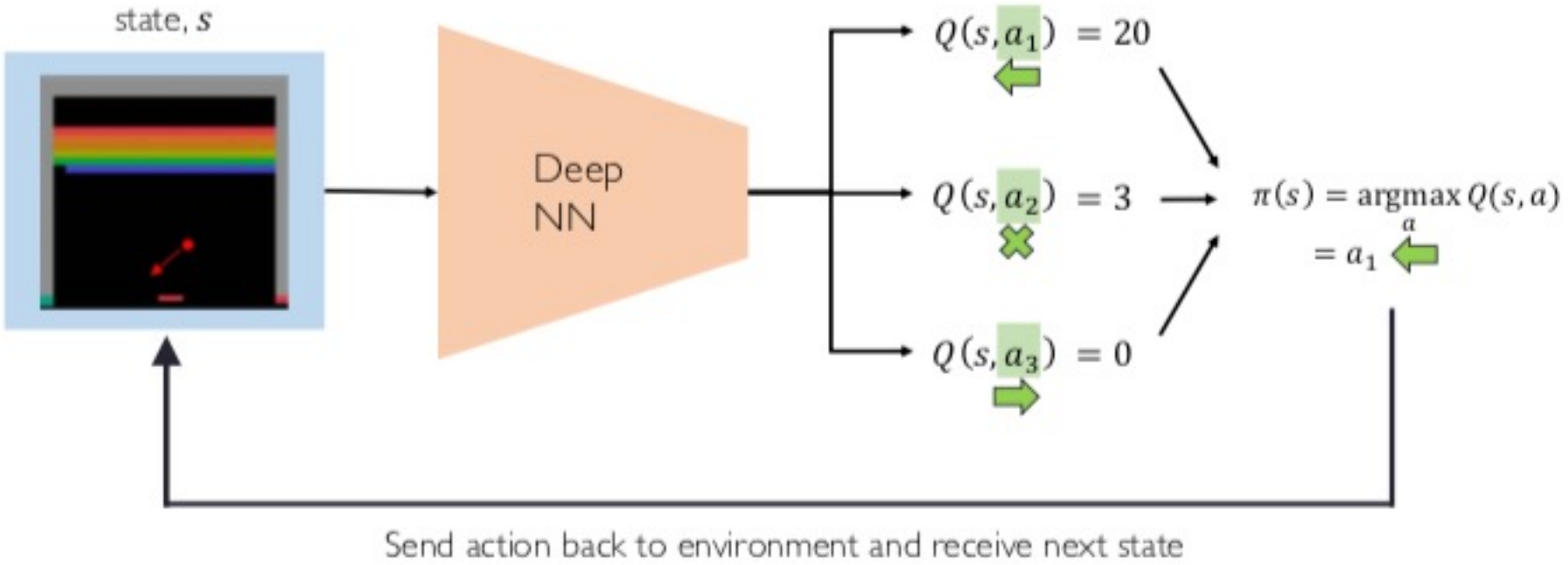


$$\mathcal{L} = \mathbb{E}\left[\left\|\overbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}^{\text{target}} - \overbrace{Q(s, a)}^{\text{predicted}}\right\|^2\right] \quad \text{Q-Loss}$$

Given the target and the prediction, we can define a loss. Use the loss to backprop and learn the network parameters.

From Q-function to Policy

Use NN to learn Q-function and then use to infer the optimal policy, $\pi(s)$



DQN without the Batch Setting

Fitted Q-Learning (Control Setting)

Use parametric model for Q function: $Q_\theta(x, u)$

Gradient ascent on θ:

$$\theta \leftarrow \theta + \alpha \left(r_t + \gamma \max_u Q_\theta(x_{t+1}, u) - Q_\theta(x_t, u_t) \right) \nabla_\theta Q_\theta(x_t, u_t)$$

learning rate $\quad \frac{d(Squared\,TD\,Error)}{dQ} \quad \frac{dQ}{d\theta}$

Q-Learning and Fitted Q-Learning

Q-Learning (Batch Setting)

Initialize $Q(x,u)$ for all states and actions

Gather trajectory data $\tau = (x_0, u_0, r_0, x_1, \dots, u_{H-1}, r_{H-1}, x_H)$

Loop:

Take a piece of experience (x_t, u_t, r_t, x_{t+1})

Update Q to correct TD error:

$$Q(x_t,u_t) \leftarrow Q(x_t,u_t) + \alpha\left(r_t + \max_u Q(x_{t+1},u) - Q(x_t,u_t)\right)$$

$t = t+1$

Fitted Q-Learning (Control Setting)

Use parametric model for Q function: $Q_\theta(x,u)$

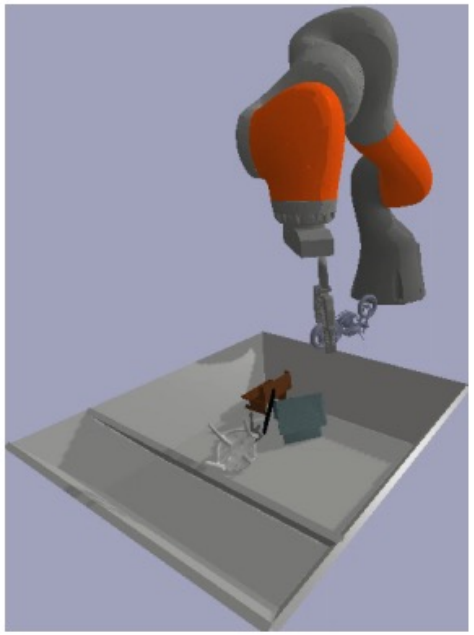
Gradient ascent on θ:

$$\theta \leftarrow \theta + \alpha\left(r_t + \gamma \max_u Q_\theta(x_{t+1},u) - Q_\theta(x_t,u_t)\right)\nabla_\theta Q_\theta(x_t,u_t)$$

learning rate $\quad \frac{d(Squared\,TD\,Error)}{dQ} \quad \frac{dQ}{d\theta}$

Robot Grasping: QT-Opt

- State: Image, gripper status, height
- The agent is given a reward of 1 at the end of an episode if it has successfully grasped an object from the bin. In all other cases the reward is 0. Positive reward if the grasp is successful.
- The agent's action comprises a gripper pose displacement, and an open/close command. The gripper pose displacement is a difference between the current pose and the desired pose.



- https://sites.google.com/view/qtopt

QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation

QT-Opt

Figure 1: Seven robots are set up to collect grasping episodes with autonomous self-supervision.

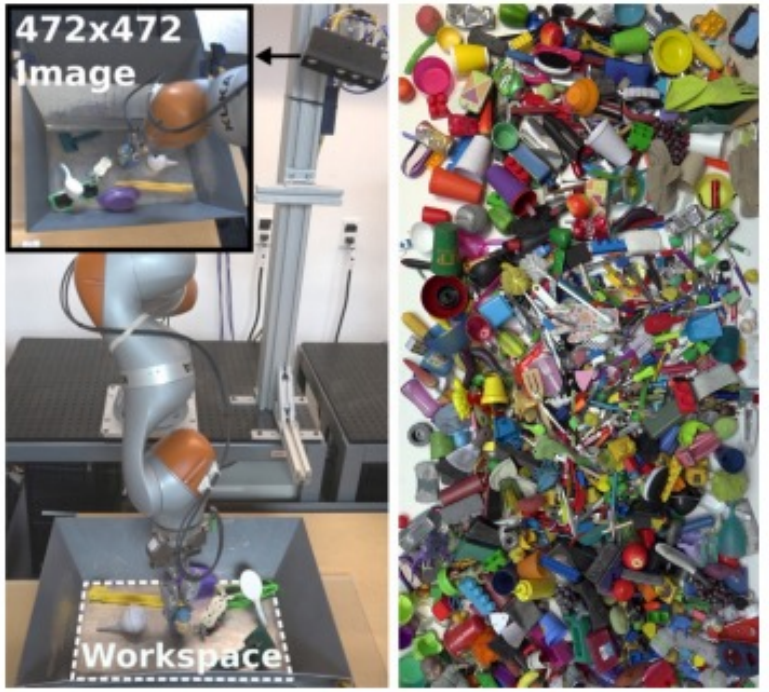
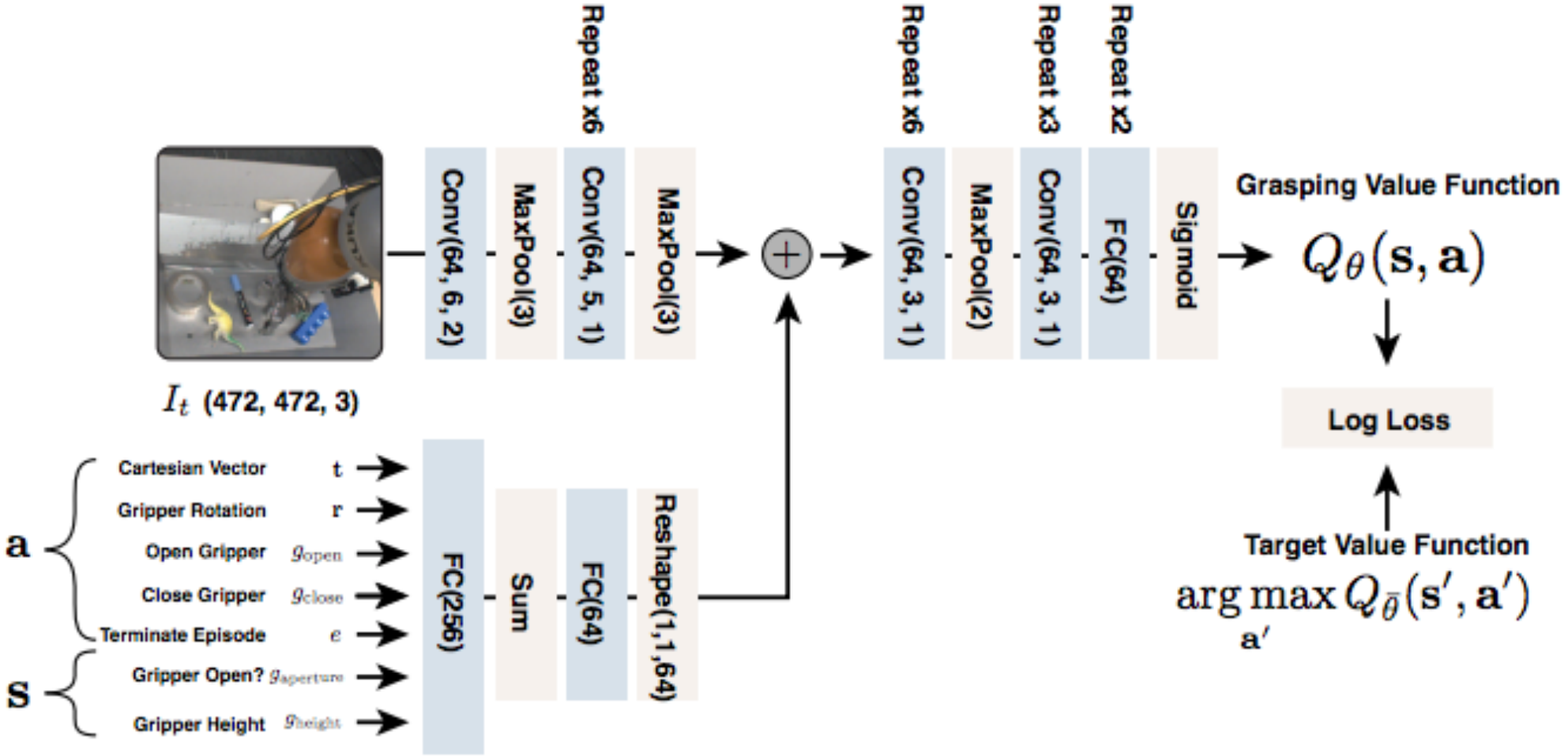


Figure 2: Close-up of a robot cell in our setup (left) and about 1000 visually and physically diverse training objects (right). Each cell (left) consists of a KUKA LBR IIWA arm with a two-finger gripper and an over-the-shoulder RGB camera.

QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation

QT-Opt



QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation

QT-Opt

QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation

QT-Opt

Figure 4: Eight grasps from the QT-Opt policy, illustrating some of the strategies discovered by our method: pregrasp manipulation (a, b), grasp readjustment (c, d), grasping dynamic objects and recovery from perturbations (e, f), and grasping in clutter (g, h). See discussion in the text and Appendix A.

Algorithm 1 Grasping control-loop

1: Pick a policy *policy*.
2: Initialize a *robot*.
3: **while** *step* $< N$ and not *terminate_episode* **do**
4: $\mathbf{s}$ = *robot*.CaptureState()
5: $\mathbf{a}$ = *policy*.SelectAction($\mathbf{s}$)
6: *robot*.ExecuteCommand($\mathbf{a}$)
7: *terminate_episode* = e {Termination action e is either learned or decided heuristically.}
8: r = *robot*.ReceiveReward()
9: emit($\mathbf{s}$, $\mathbf{a}$, r)
10: **end while**

QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation

Catastrophic Forgetting

- Problem: Neural network regression assumes that training data are “IID”
  - The data are drawn Independently from the Identical Distribution.

- Temporal Correlations (*independence assumption*)
  - When a learning agent is moving through an environment, the sequence of states it encounters will be temporally correlated.

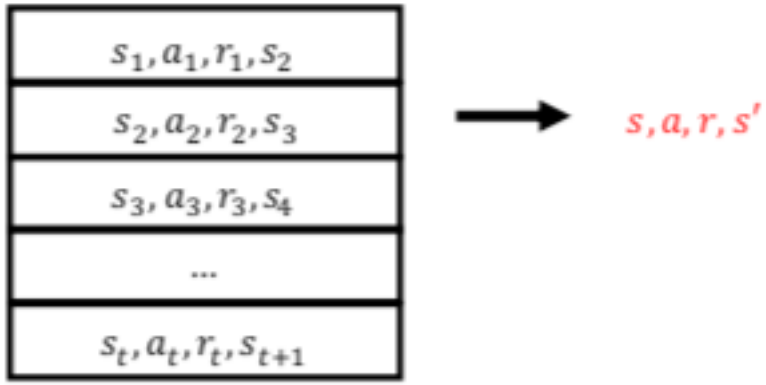
- Day and Night Robot Example (*identically distributed*)
  - Imagine a robot that spends 12 hours in a dark environment and 12 hour in a light one”.
  - Can mean that while it is in the dark, the neural-network weight-updates will make the Q function “forget” the value function for when it’s light.
  - Example from Leslie Kaelbling and Lozano-Perez.

- Use of Experience Replay
  - To avoid catastrophic forgetting
  - To break temporal correlations/ remove correlations

Experience Replay

- Solution: use experience replay, where we save our (s_t, a_t, r_t, s_{t+1}) experiences in a replay buffer.
  - Whenever we take a step in the world, we add the (s_t,a_t,r_t,s_{t+1}) to the replay buffer and use it to do a Q-learning update.
  - Then we also randomly select some number of tuples from the replay buffer, and do Q-learning updates based on them, as well.

Maintain a replay buffer



- To perform experience replay, repeat the following:
  - $(s, a, r, s') \sim \mathcal{D}$: sample an experience tuple from the dataset
  - Compute the target value for the sampled s: $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w})$
  - Use stochastic gradient descent to update the network weights

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})$$

Ways to prioritize updates

- Random
  - Select the tuple at random.
- Proportional Stochastic Prioritization
  - Order of replaying updates could help speed learning

- Let i be the index of the ith tuple of experience (s_i, a_i, r_i, s_{i+1})
- Sample tuples for update using priority function
- Priority of a tuple i is proportional to DQN error

$$p_i = |r + \gamma \max_a Q(s_{i+1}, a', \mathbf{w}') - Q(s_i, a_i, \mathbf{w})|$$

- Update p_i every update
- p_i for new tuples is set to 0

$$p(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

QT-Opt

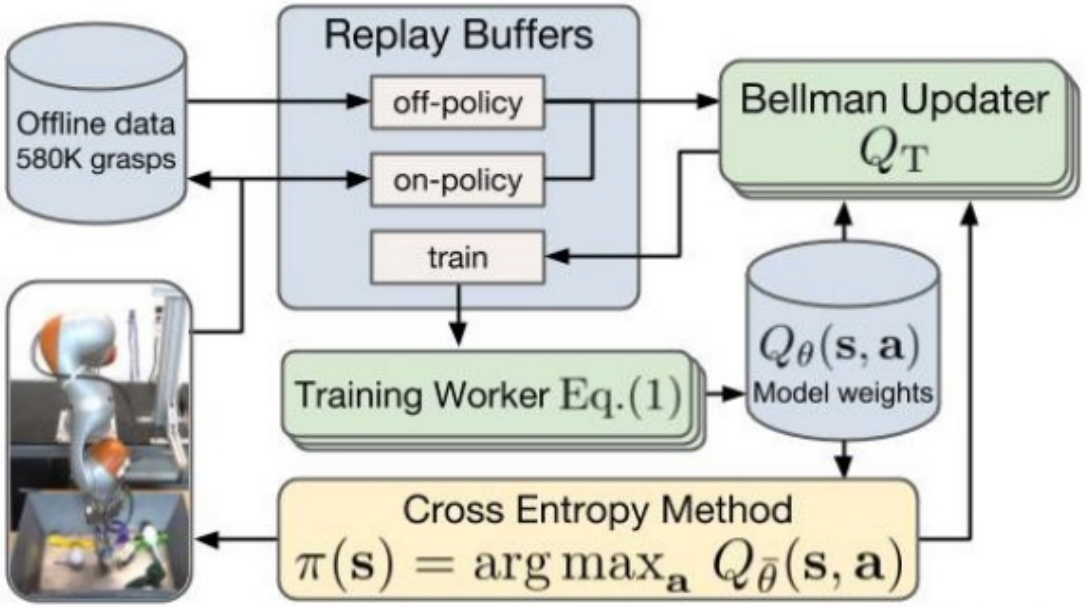
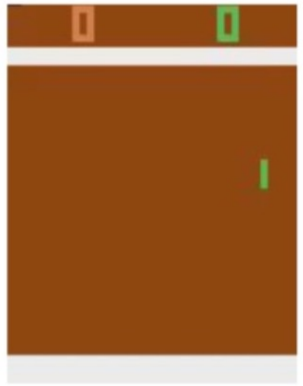


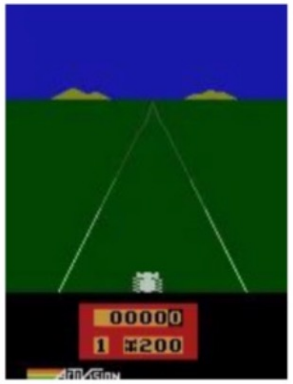
Figure 3: Our distributed RL infrastructure for QT-Opt (see Sec. 4.2). State-action-reward tuples are loaded from an offline data stored and pushed from online real robot collection (see Sec. 5). Bellman update jobs sample transitions and generate training examples, while training workers update the Q-function parameters.

QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation

DQN successes in game play

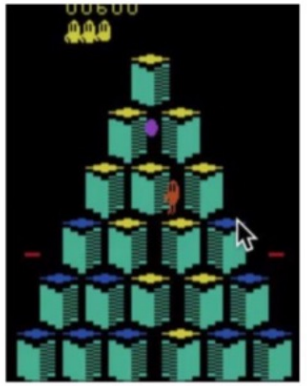


Pong



Enduro

Beamrider

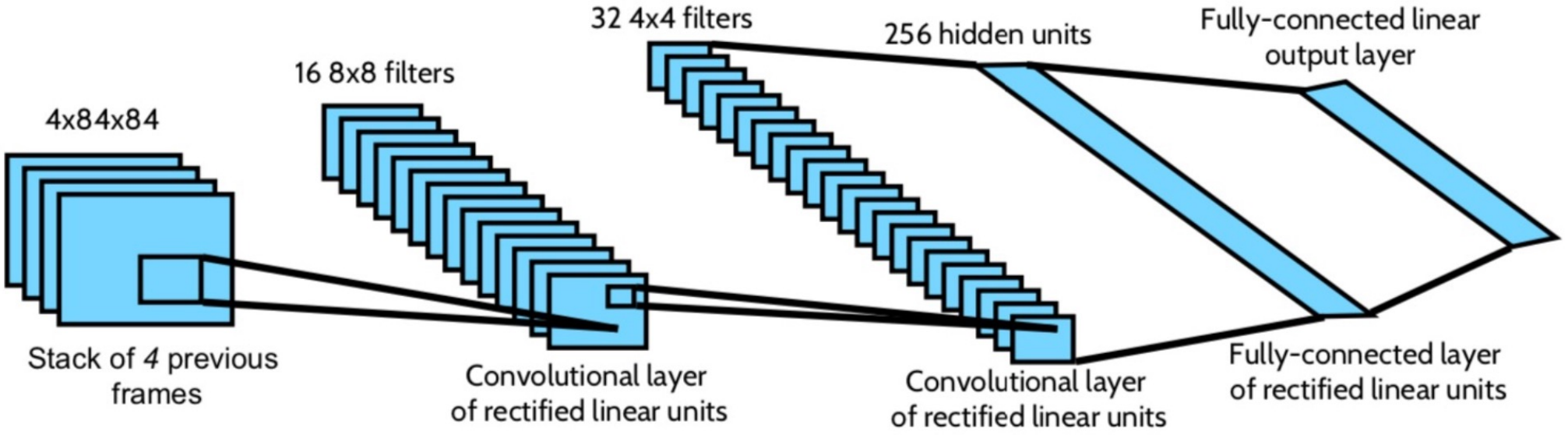


Q*bert

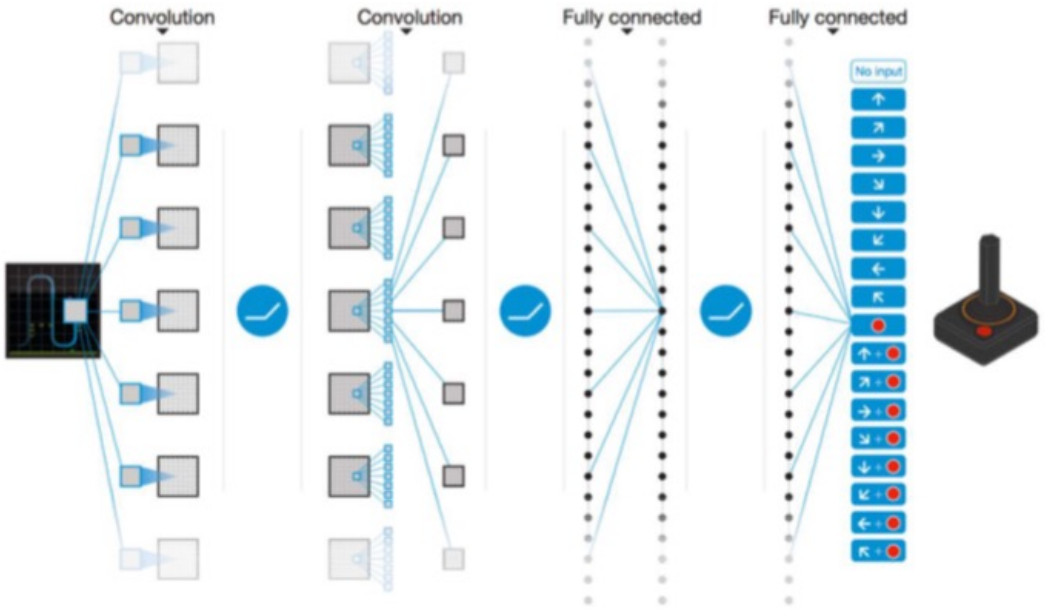
- 49 ATARI 2600 games.
- From pixels to actions.
- The change in score is the reward.
- Same algorithm.
- Same function approximator, w/ 3M free parameters.
- Same hyperparameters.
- Roughly human-level performance on 29 out of 49 games.

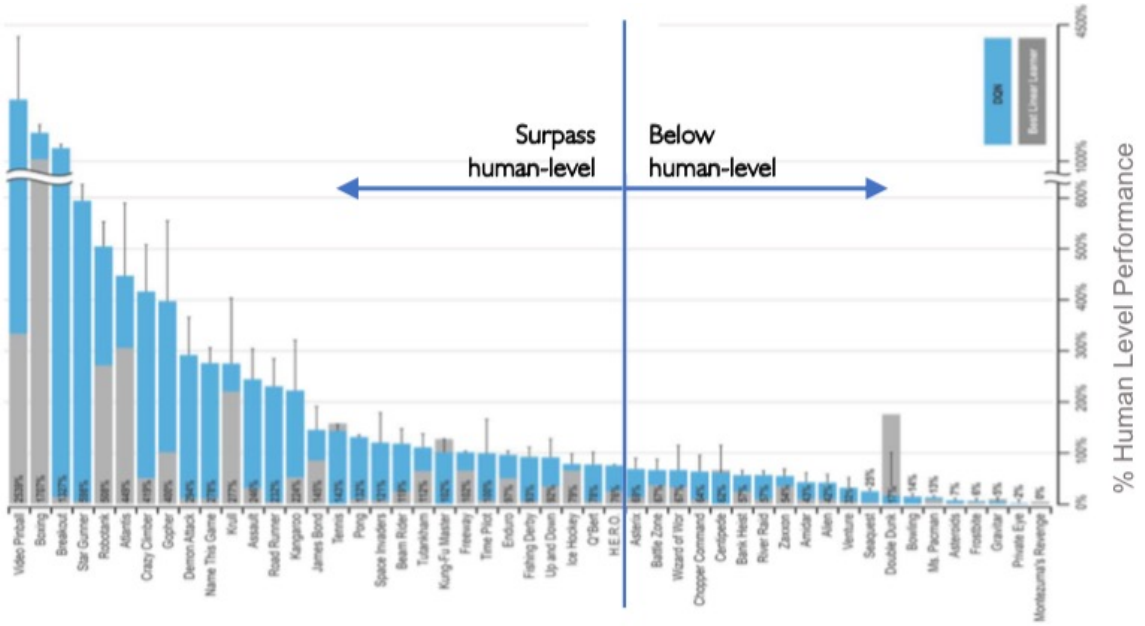
DQN successes in game play

- Convolutional neural network architecture:
  - History of frames as input.
  - One output per action - expected reward for that action $Q(s, a)$.
  - Final results used a slightly bigger network (3 convolutional + 1 fully-connected hidden layers).



DQN successes in game play





Use of CNNs to extract a state representation. Output is a movement of the agent in the grid.

Deep Q-Learning with Experience Replay

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

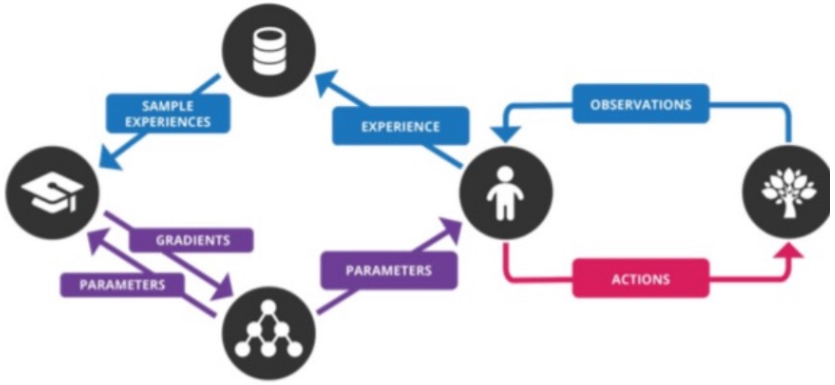
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

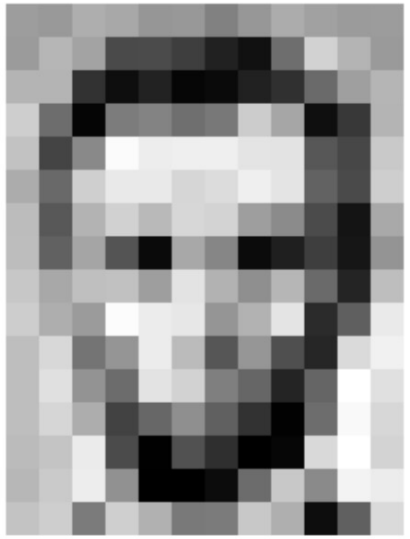
End For



DeepMind

Aside: Representation Learning

- Representation learning: Mapping raw observations to features and structures from which the mapping to actions or to semantic labels is easier to infer.
- Observations (images in this example) are to be represented in some way for the agent to use for decision making.

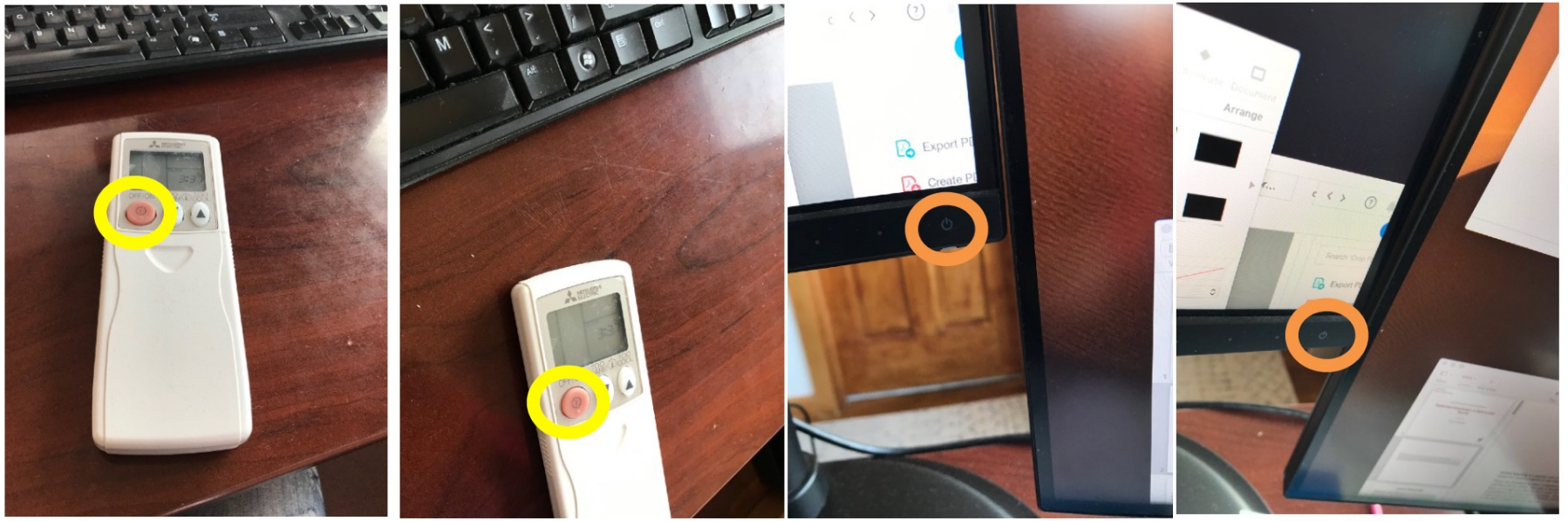


157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

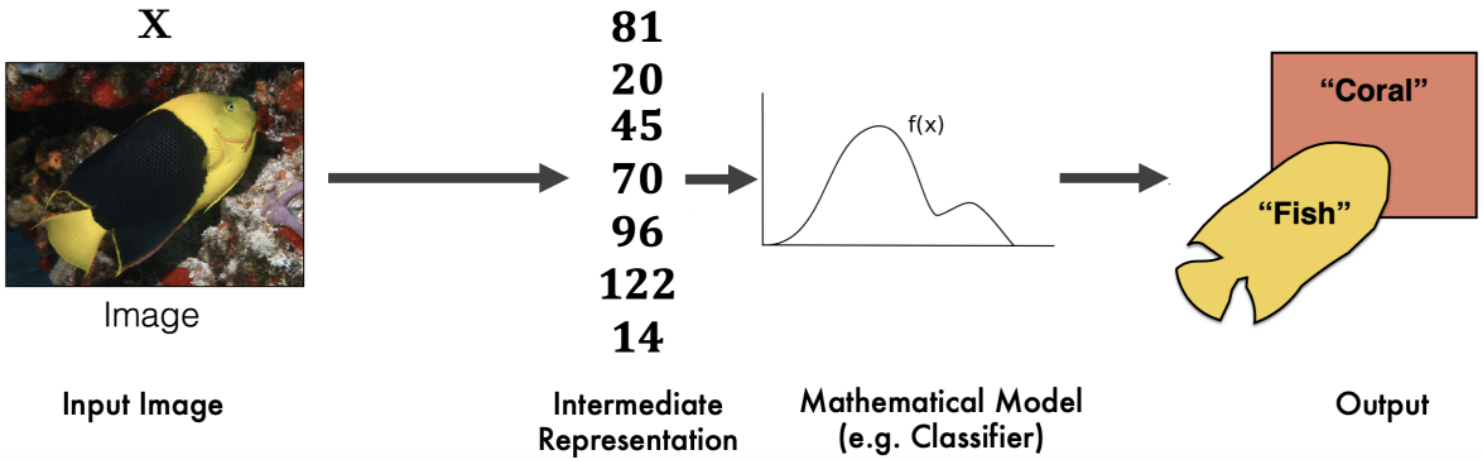
Representation Learning aids Learning to Act

- The raw representation of images has very different pixel values.
- However, the actions required to achieve the goal of switching on the device are similar.



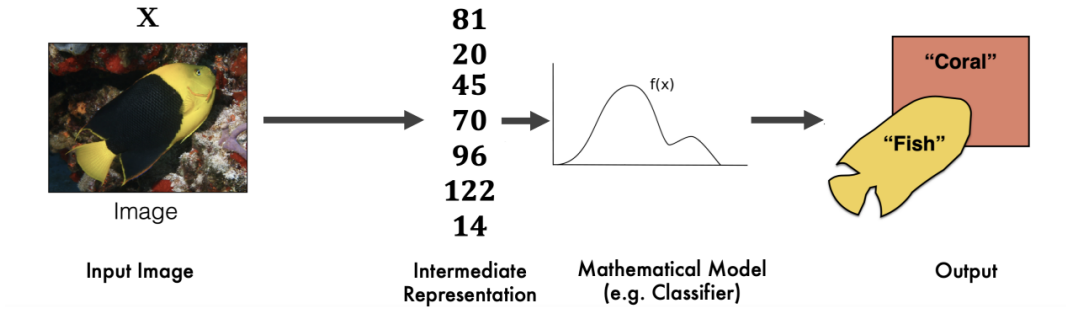
Representation Learning aids Learning to Act

- The raw representation of images has very different pixel values.
  - The actions required to achieve the goal of switching on the device are similar.
- Visual perception is instrumental to learning to act
  - The representation transforming raw pixels to action-relevant feature vectors and structures.

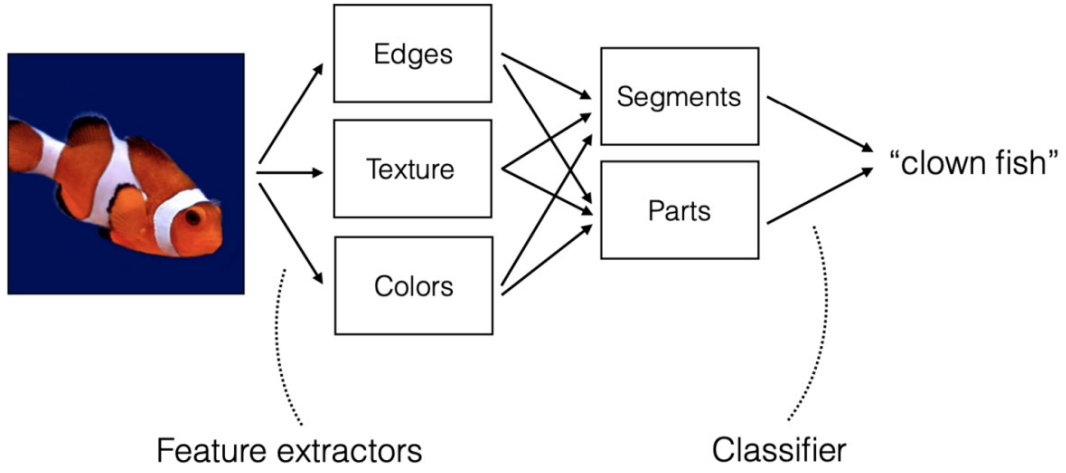


Representations for (Visual) data

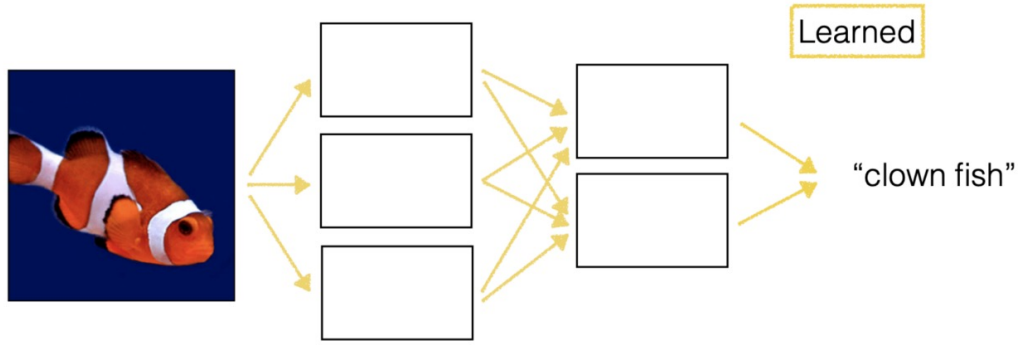
Representation of data for a downstream recognition task.



Traditional paradigm: features are hand specified



Modern Paradigm: Features are learned.



Representations can fuse many modalities

Making Sense of Vision and Touch: Self-Supervised Learning of Multimodal Representations for Contact-Rich Tasks

Michelle A. Lee*, Yuke Zhu*, Krishnan Srinivasan, Parth Shah,
Silvio Savarese, Li Fei-Fei, Animesh Garg, Jeannette Bohg

- Considers: peg-in-hole insertion tasks.
- The robot both observes the task being performed and the force interaction measurements during the task.
- The goal is to learn a policy so that the peg is inserted in the hole for an assembly task.
- Paper: https://rpl.cs.utexas.edu/publications/papers/lee-icra19-making.pdf

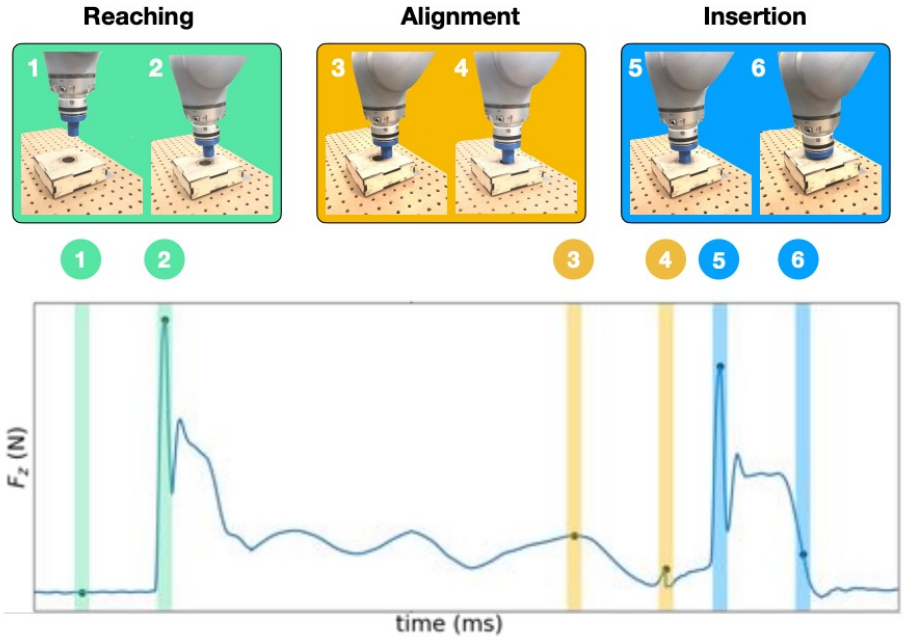


Fig. 1: Force sensor readings in the z-axis (height) and visual observations are shown with corresponding stages of the peg insertion task. When reaching for the box, the force reading transitions from (1) the arm being in free space to (2) being in contact with the box. While aligning the peg, the forces capture the dynamics of contact as the peg slides on the box surface (3, 4). Finally, in the insertion stage, the forces peak as the robot attempts to insert the peg at the

Representations can fuse many modalities

Core idea is to learn a multi-modal representation from visual and touch data.

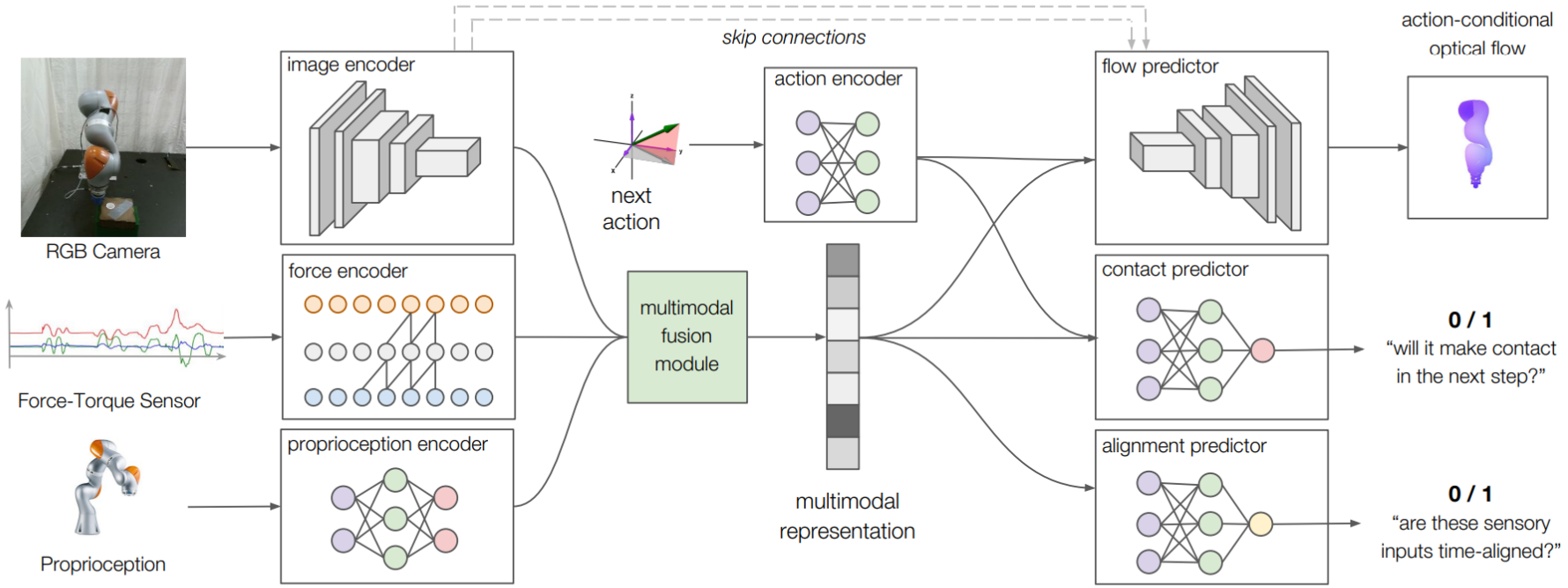


Fig. 2: Neural network architecture for multimodal representation learning with self-supervision. The network takes data from three different sensors as input: RGB images, F/T readings over a 32ms window, and end-effector position and velocity. It encodes and fuses this data into a multimodal representation based on which controllers for contact-rich manipulation can be learned. This representation learning network is trained end-to-end through self-supervision.

Limitations

- Works well when the the action space is discrete and small. Need to optimize over actions.
  - Hard to apply to continuous action spaces
- Policy is deterministically computed from the Q function by maximizing the rewards. I.e., there is one policy that is output.
  - Cannot model stochastic policies (distribution over actions for a state)
- The optimal Q-function may be harder to learn in comparison to the optimal policy
  - Can we learn the policy directly?