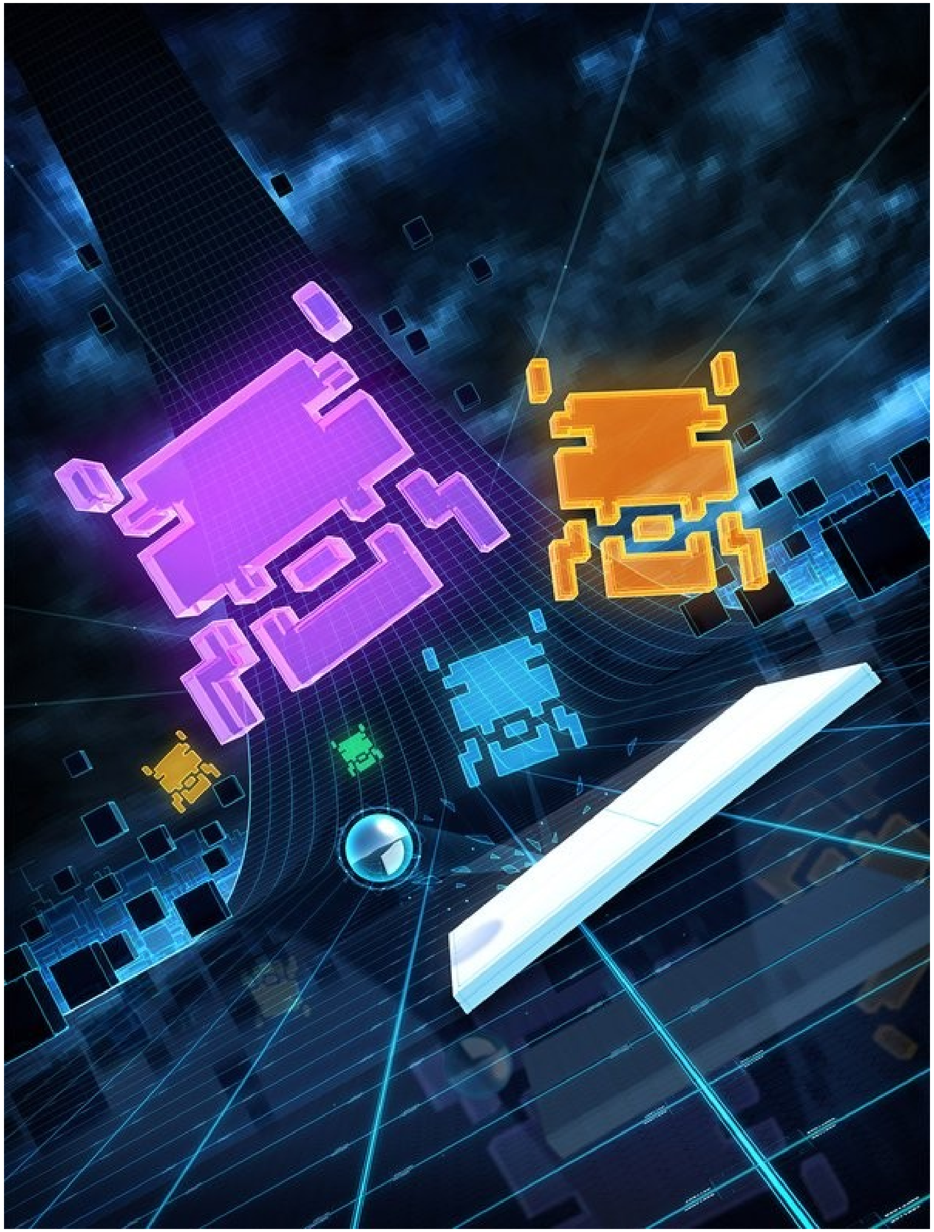
RL + DL

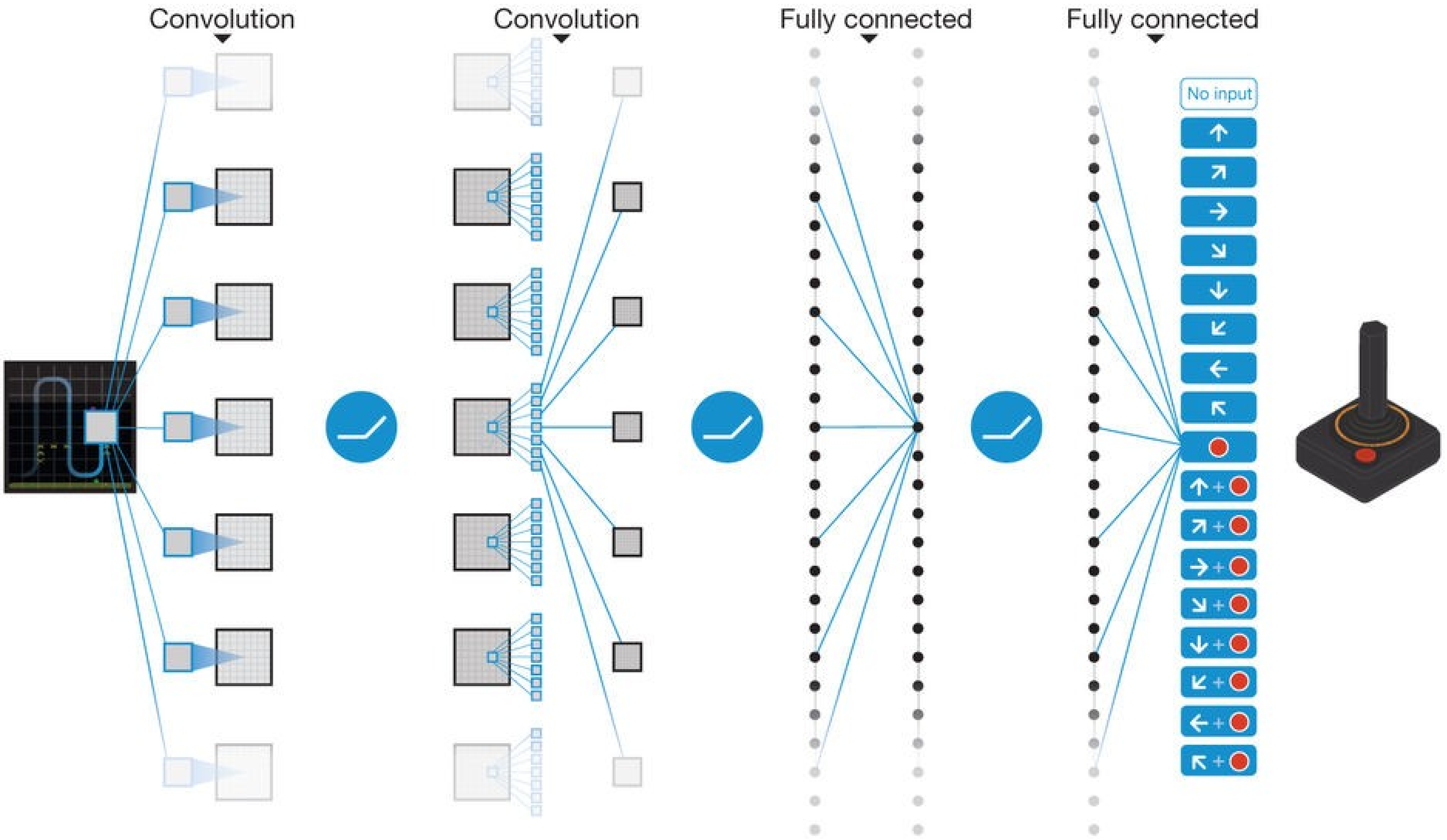
Rob Platt
Northeastern University

Some slides from David Silver, *Deep RL Tutorial*

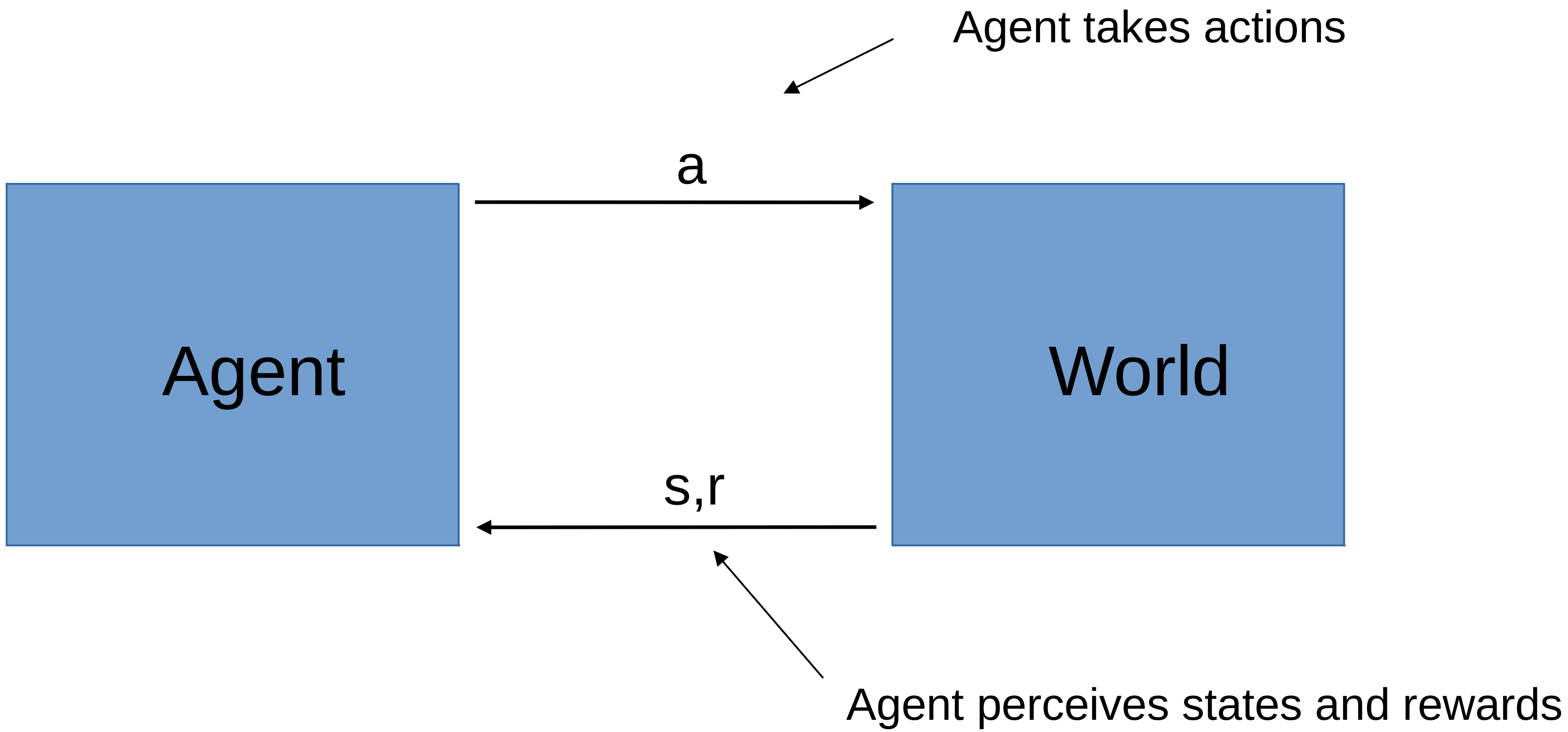
Deep RL in an exciting recent development



Deep RL in an exciting recent development

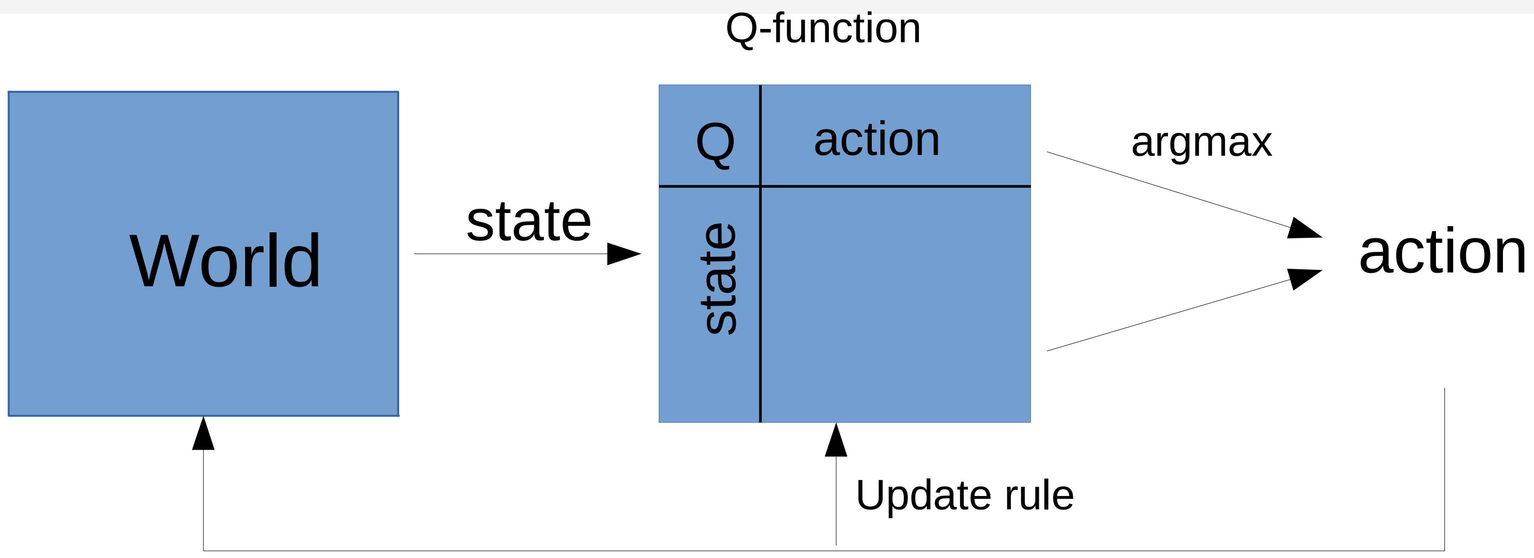


Remember the RL scenario...

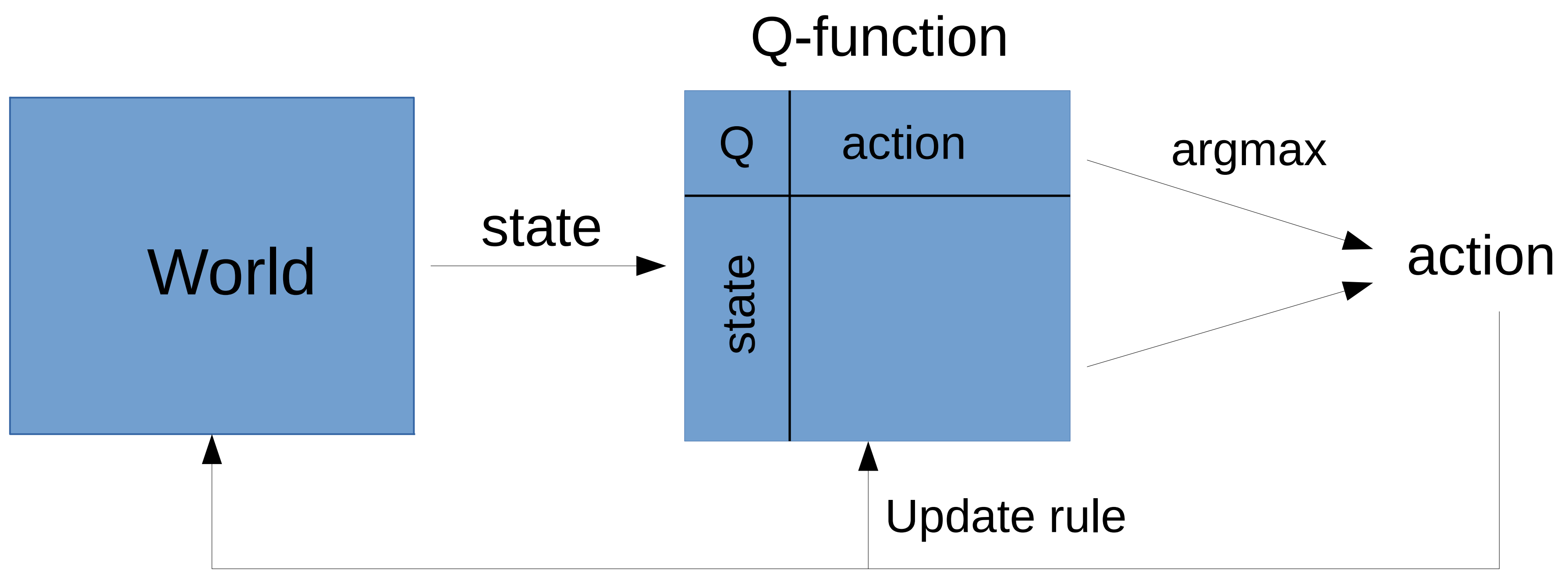


Tabular RL

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$
    until S is terminal

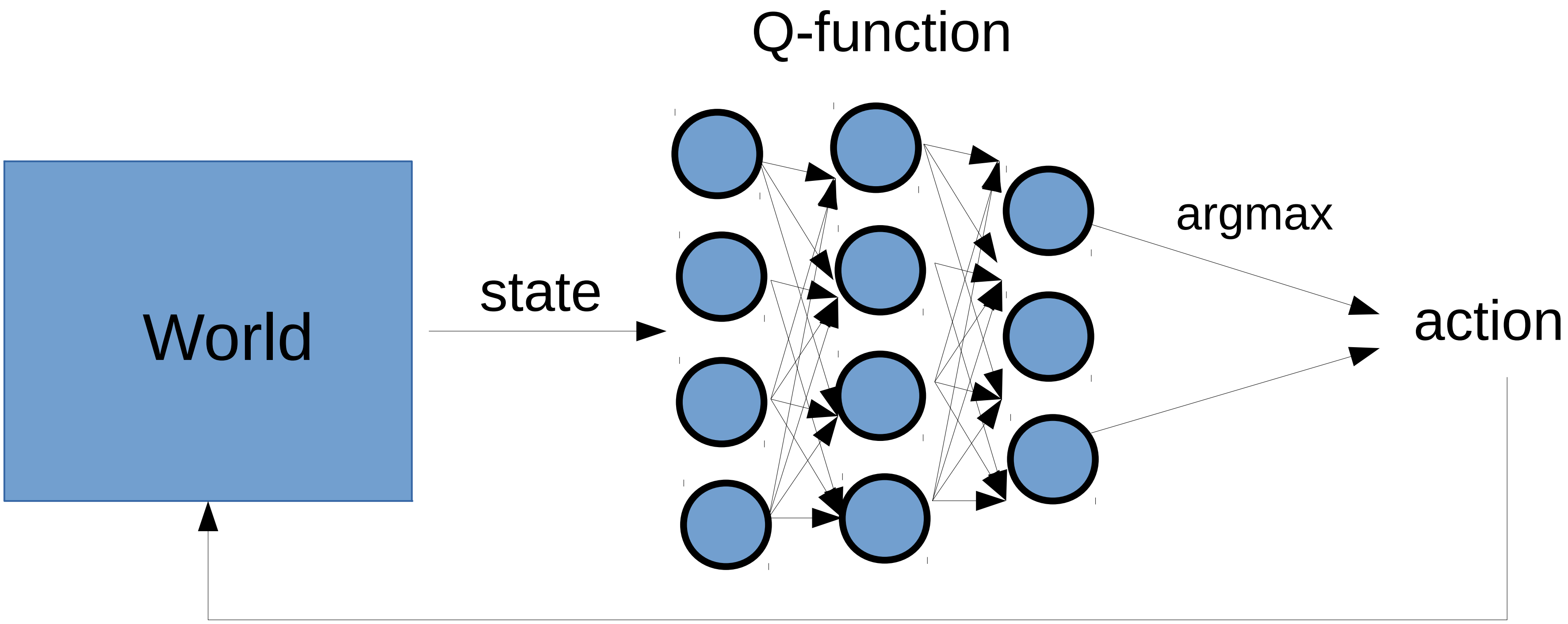


Tabular RL

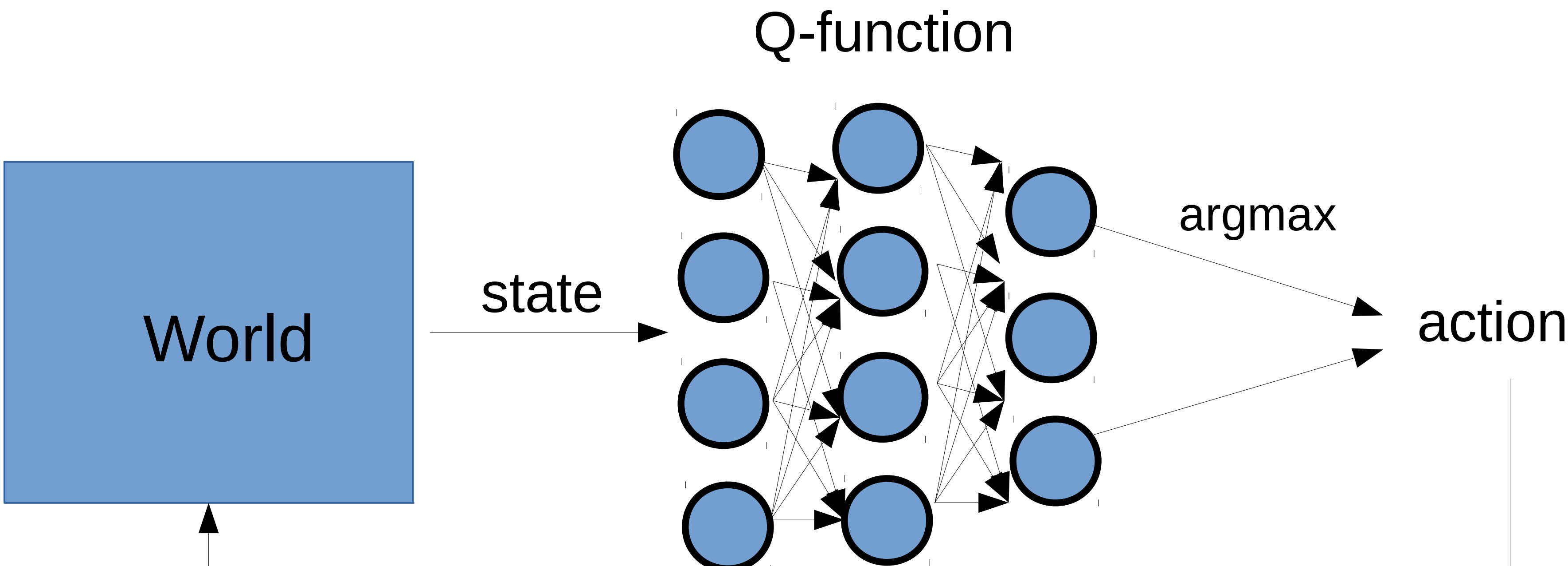


Q-function
World
state
Q
action
state
argmax
action
Update rule

Deep RL

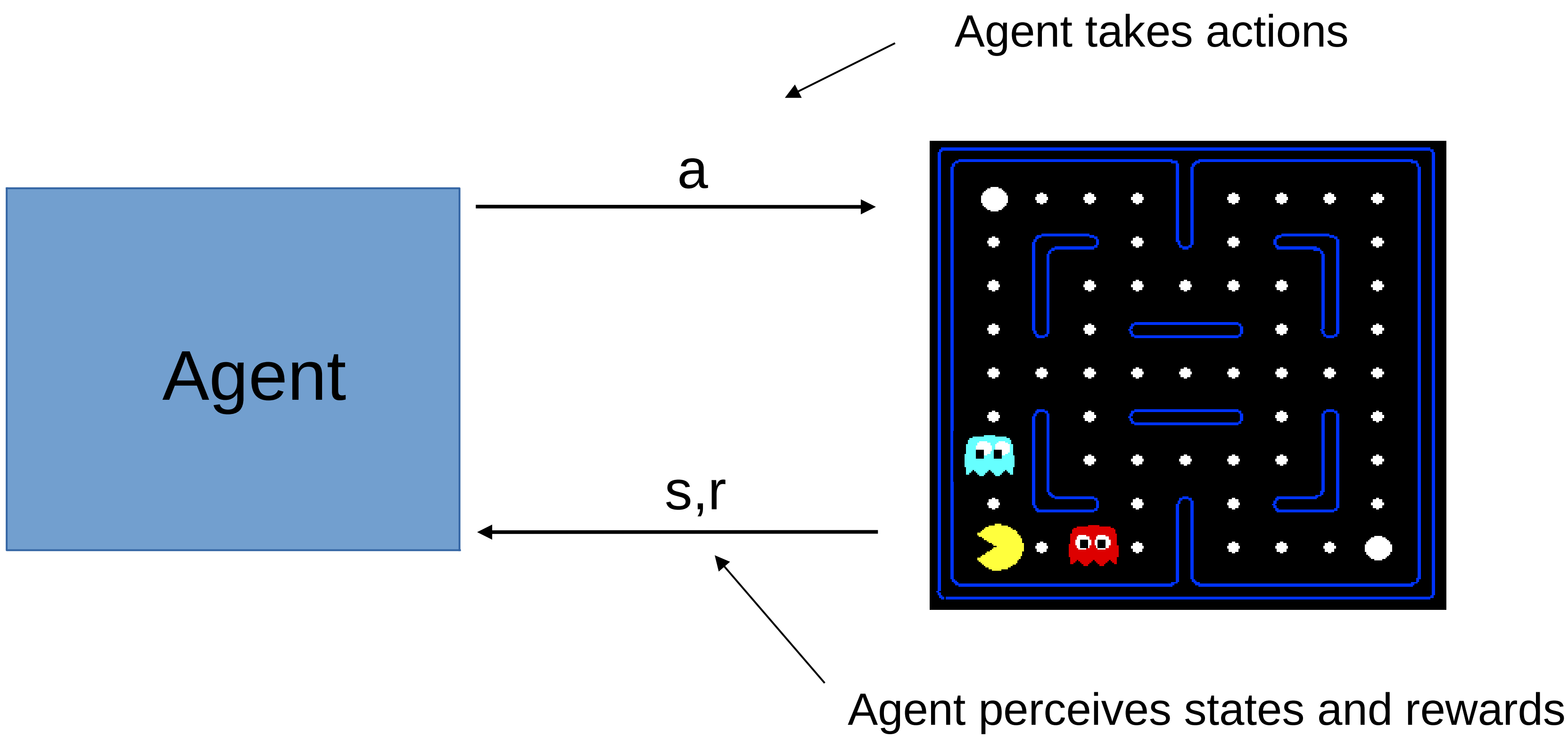


Deep RL



But, why would we want to do this?

Where does “state” come from?

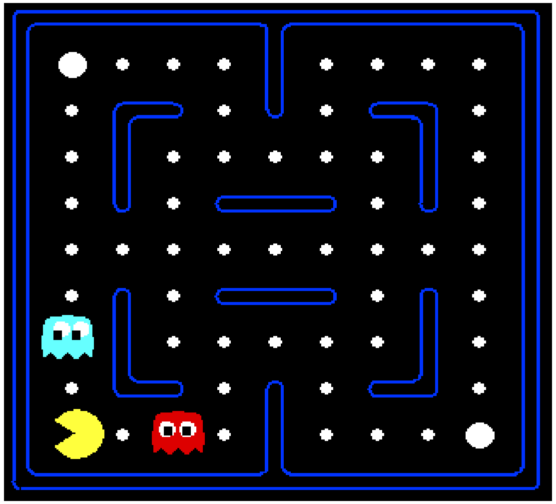


Earlier, we dodged this question: “it’s part of the MDP problem statement”

But, that’s a cop out. How do we get state?

Typically can’t use “raw” sensor data as state w/ a tabular Q-function
– it’s too big (e.g. pacman has something like 2^(num pellets) + … states)

Linear function approximation?

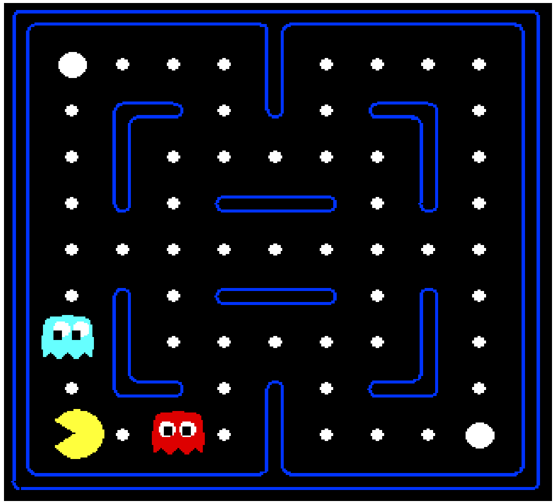


In the RL lecture, we suggested using Linear Function approximation:

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

Q-function encoded by the weights

Linear function approximation?



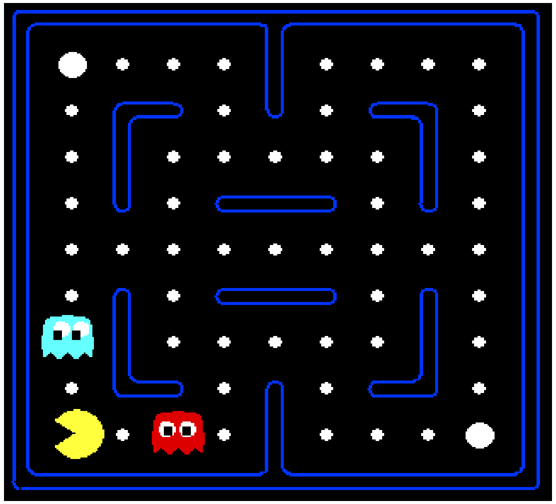
In the RL lecture, we suggested using Linear Function approximation:

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

Q-function encoded by the weights

But, this STILL requires a human designer to specify the features!

Linear function approximation?



In the RL ximation:

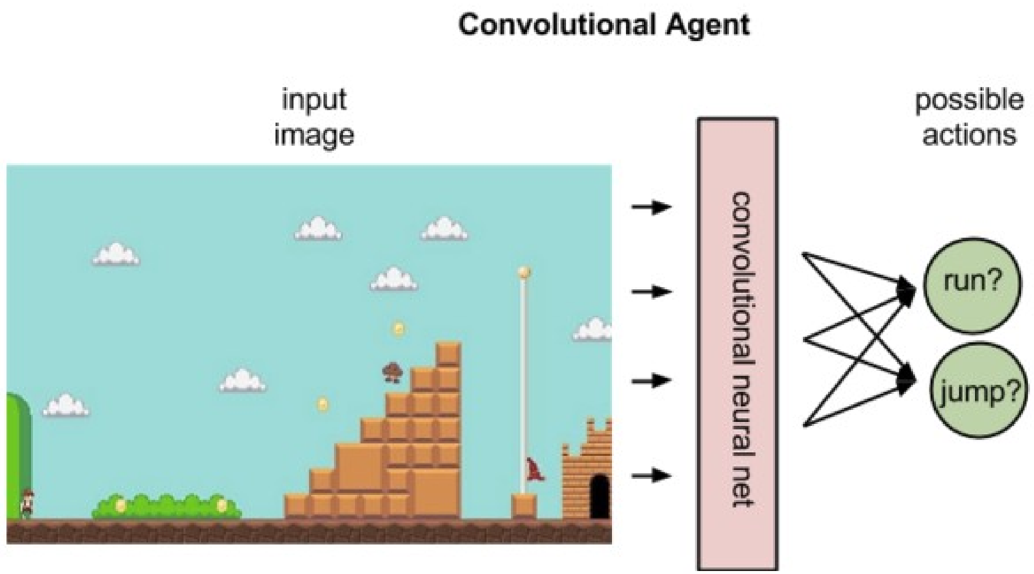
$$Q(\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad a)$$

Is it possible to do RL WITHOUT
hand-coding either states or features?

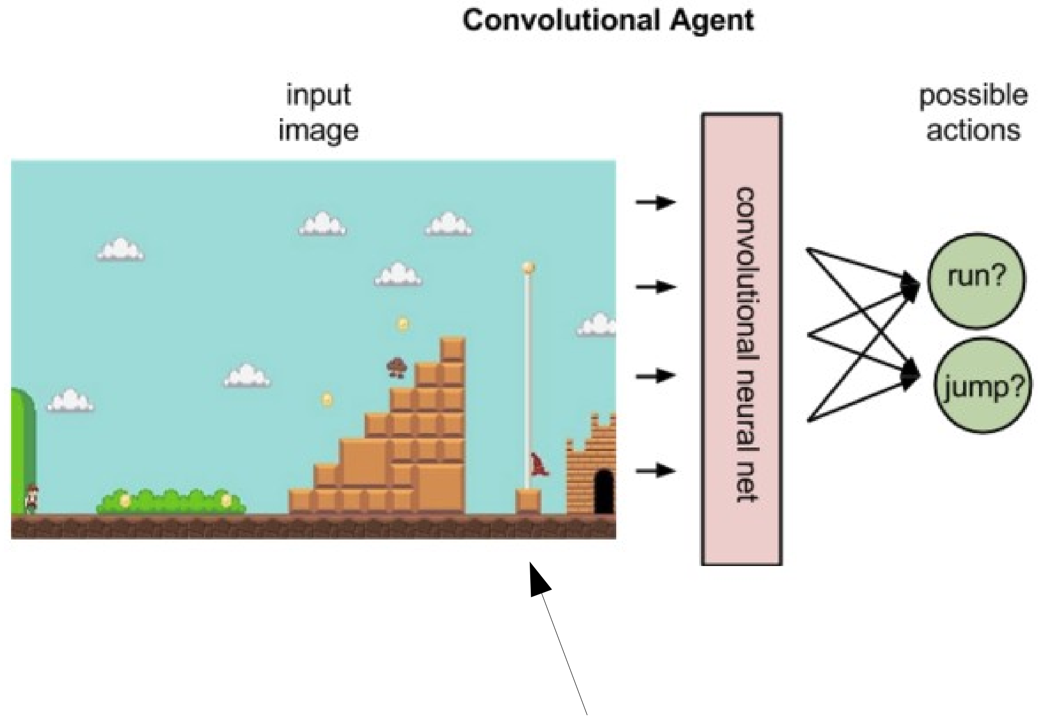
Q-functio

But, this ures!

Deep Q Learning



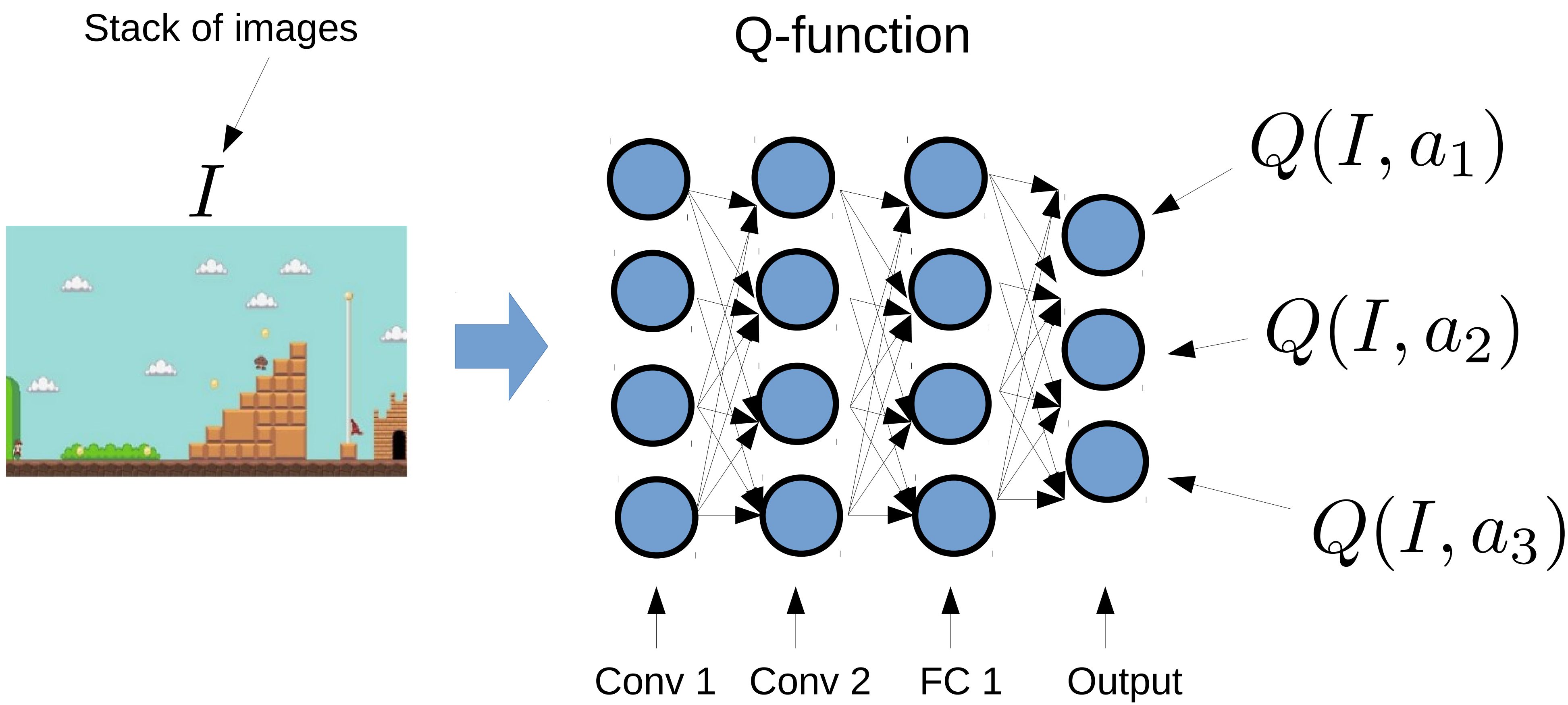
Deep Q Learning



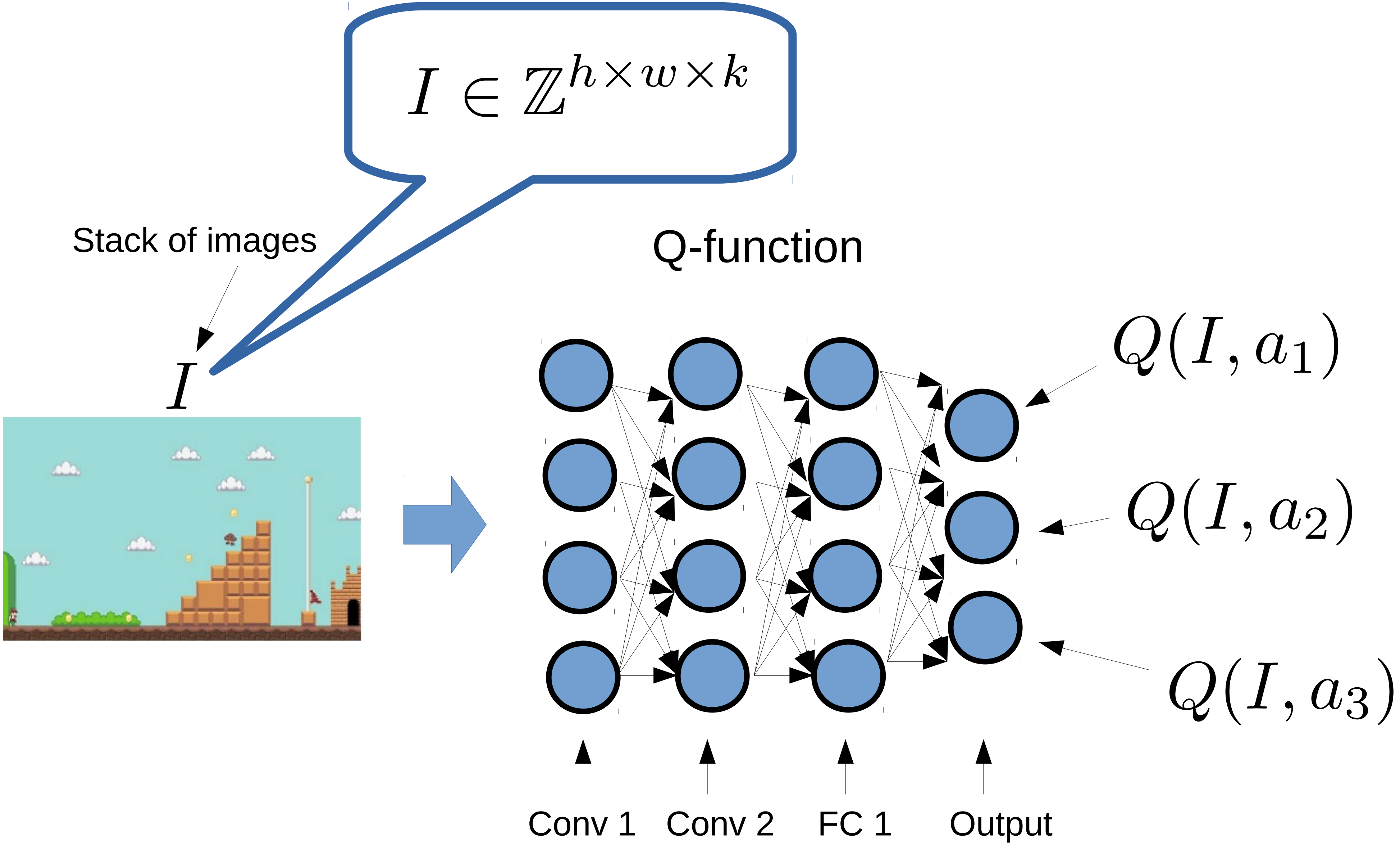
Instead of state, we have an image
- in practice, it could be a history of the *k* most recent images stacked as a single *k*-channel image

Hopefully this new image representation is Markov…
- in some domains, it might not be!

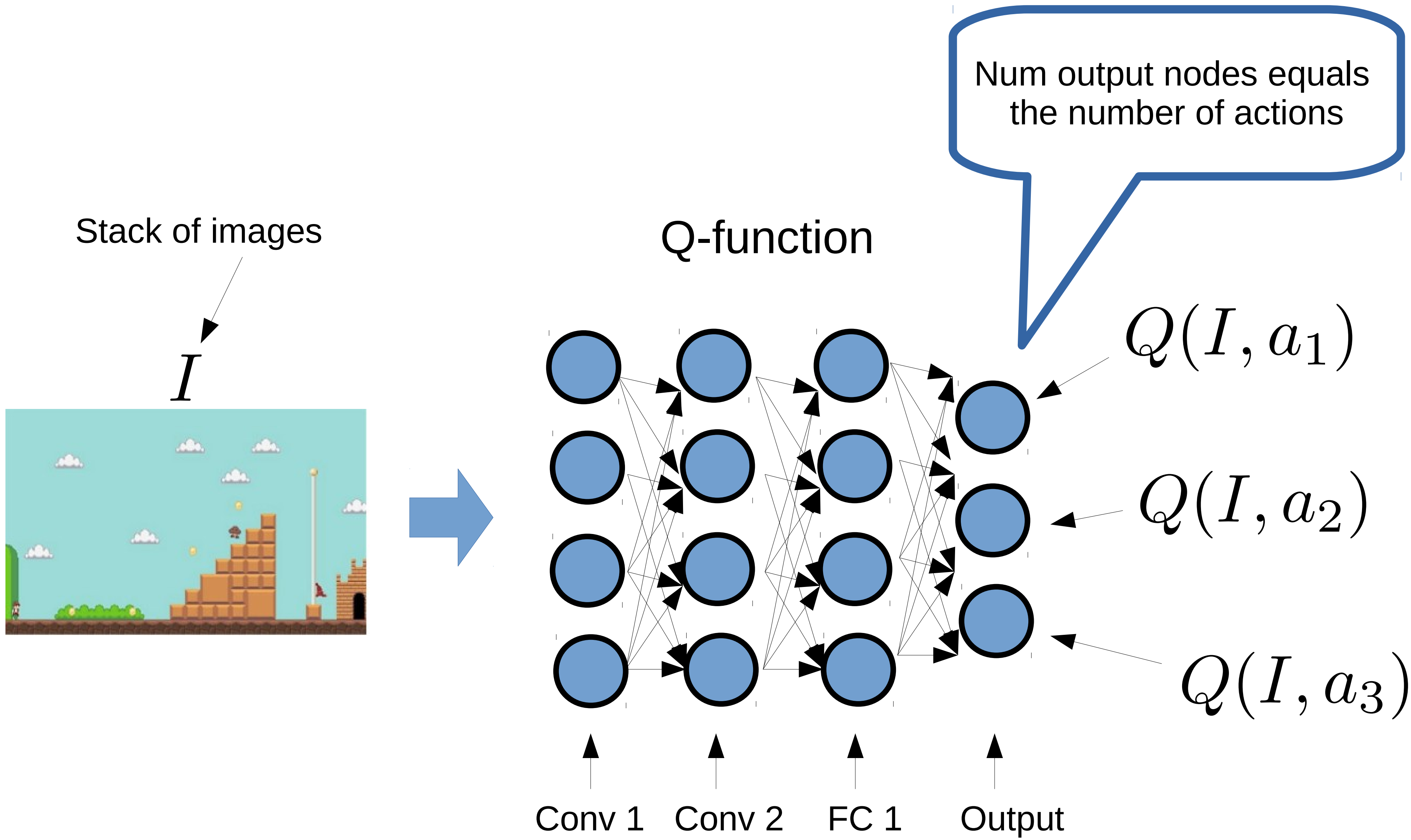
Network structure



Network structure



Network structure



Updating the Q-function

Here's the standard Q-learning update equation:

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

Updating the Q-function

Here's the standard Q-learning update equation:

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

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$
    until S is terminal

Updating the Q-function

Here's the standard Q-learning update equation:

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

Rewriting:

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') \right]$$

Updating the Q-function

Here's the standard Q-learning update equation:

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

Rewriting:

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha \underbrace{\left[r + \gamma \max_{a'} Q(s',a') \right]}_{\text{let's call this the “target”}}$$

This equation adjusts Q(s,a) in the direction of the target

Updating the Q-function

Here's the standard Q-learning update equation:

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

Rewriting:

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha \underbrace{\left[r + \gamma \max_{a'} Q(s',a') \right]}_{\text{let's call this the "target"}}$$

This equation adjusts Q(s,a) in the direction of the target

We're going to accomplish this same thing in
a different way using neural networks...

Remember how we train a neural network...

Given a dataset: $D = \{(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)\}$

Define loss function: $L(x^i, y^i; w, b) = \frac{1}{2}(y^i - f(w^T x^i + b))^2$

Adjust *w, b* so as to minimize: $\sum_{(x^i, y^i) \in D} L(x^i, y^i; w, b)$

<u>Do gradient descent on dataset:</u>

1. repeat
2. $$w \leftarrow w - \alpha \frac{1}{n} \sum_{(x^i, y^i) \in D} \nabla_w L(x^i, y^i; w, b)$$
3. $$b \leftarrow b - \alpha \frac{1}{n} \sum_{(x^i, y^i) \in D} \nabla_b L(x^i, y^i; w, b)$$
4. until converged

Updating the Q-network

Use this loss function:

$$L(s, a, s'; w) = \frac{1}{2}\left(r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a)\right)^2$$

Updating the Q-network

Use this loss function:

$$L(s, a, s'; w) = \frac{1}{2}\left(r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a)\right)^2$$

Notice that Q is now parameterized by the weights, *w*

Updating the Q-network

Use this loss function:

I'm including the bias in the weights

$$L(s,a,s';w) = \frac{1}{2}\left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a)\right)^2$$

Updating the Q-network

Use this loss function:

$$L(s, a, s'; w) = \frac{1}{2}\left(\overset{\text{target}}{\boxed{r + \gamma \max_{a'} Q_w(s', a')}} - Q_w(s, a)\right)^2$$

Updating the Q-network

Use this loss function:

$$L(s,a,s';w) = \frac{1}{2}\Big(\overbrace{\boxed{r + \gamma \max_{a'} Q_w(s',a')}}^{\text{target}} - Q_w(s,a)\Big)^2$$

Notice that the target is a function of the weights
– this is a problem b/c targets are non-stationary

Updating the Q-network

Use this loss function:

$$L(s,a,s';w) = \frac{1}{2}\Big(\underbrace{r + \gamma \max_{a'} Q_w(s',a')}_{\text{target}} - Q_w(s,a)\Big)^2$$

Notice that the target is a function of the weights
– this is a problem b/c targets are non-stationary

So… when we do gradient descent, we're
going to hold the targets constant

Now we can do gradient descent!

Deep Q Learning, no replay buffers

Initialize Q(*s,a;w*) with random weights
Repeat (for each episode):
    Initialize *s*
    Repeat (for each step of the episode):
        Choose *a* from *s* using policy derived from Q (e.g. e-greedy)
        Take action *a*, observe *r, s'*

$$w \leftarrow w - \alpha \nabla_w L(s, a, s'; w)$$

$$s \leftarrow s'$$

    Until *s* is terminal

Where:

$$\nabla_w L(s, a, s'; w) \doteq - \left(r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a) \right) \nabla_w Q_w(s, a)$$

Deep Q Learning, no replay buffers

Initialize Q(*s,a;w*) with random weights
Repeat (for each episode):
  Initialize *s*
  Repeat (for each step of the episode):
    Choose *a* from *s* using policy derived from Q (e.g. e-greedy)
    Take action *a*, observe *r, s'*

$$\boxed{\begin{aligned} & w \leftarrow w - \alpha \nabla_w L(s, a, s'; w) \\ & s \leftarrow s' \end{aligned}}$$

  Until *s* is terminal

This is all that changed relative to standard q-learning

Where:

$$\nabla_w L(s, a, s'; w) \doteq -\left(r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a)\right) \nabla_w Q_w(s, a)$$

Example: 4x4 frozen lake env

Get to the goal (G)
Don't fall in a hole (H)

```
SFFF
FHFH
FFFH
HFFG
  (Left)
```

```
steps: 11582, episodes: 770, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.0873818397522
steps: 11609, episodes: 771, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.0872020721436
steps: 11652, episodes: 772, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.138998985291
steps: 11672, episodes: 773, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.0649240016937
steps: 11689, episodes: 774, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.0546970367432
steps: 11697, episodes: 775, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.0260739326477
steps: 11731, episodes: 776, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.110991954803
steps: 11773, episodes: 777, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.135339975357
steps: 11798, episodes: 778, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.0810689926147
steps: 11818, episodes: 779, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.0643260478973
steps: 11870, episodes: 780, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.169064044952
steps: 11906, episodes: 781, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.117113113403
steps: 11992, episodes: 782, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.279519796371
steps: 12064, episodes: 783, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.234206199646
steps: 12090, episodes: 784, mean 100 episode reward: 0.7, % time spent exploring: 2, time elapsed: 0.0835938453674
steps: 12137, episodes: 785, mean 100 episode reward: 0.7, % time spent exploring: 2, time elapsed: 0.150979042053
steps: 12185, episodes: 786, mean 100 episode reward: 0.7, % time spent exploring: 2, time elapsed: 0.155304908752
steps: 12245, episodes: 787, mean 100 episode reward: 0.7, % time spent exploring: 2, time elapsed: 0.194122076035
steps: 12277, episodes: 788, mean 100 episode reward: 0.7, % time spent exploring: 2, time elapsed: 0.102608919144
steps: 12293, episodes: 789, mean 100 episode reward: 0.7, % time spent exploring: 2, time elapsed: 0.0520431995392
```

Demo!

Experience replay

Deep learning typically assumes independent, identically distributed (IID) training data

Experience replay

Deep learning typically assumes independent, identically distributed (IID) training data

But is this true in the deep RL scenario?

Initialize Q(*s,a;w*) with random weights
Repeat (for each episode):
    Initialize *s*
    Repeat (for each step of the episode):
        Choose *a* from *s* using policy derived from Q (e.g. e-greedy)
        Take action *a*, observe *r, s'*

$$w \leftarrow w - \alpha \nabla_w L(s, a, s'; w)$$

$$s \leftarrow s'$$

    Until *s* is terminal

Experience replay

Deep learning typically assumes independent, identically distributed (IID) training data

But is this true in the deep RL scenario?

Initialize Q($s,a;w$) with random weights
Repeat (for each episode):
    Initialize s
    Repeat (for each step of the episode):
        Choose a from s using policy derived from Q (e.g. e-greedy)
        Take action a, observe r, s'

<u>Our solution:</u> buffer experiences and then "replay" them during training

Deep Q Learning WITH replay buffers

Initialize Q(*s,a;w*) with random weights

$D \leftarrow \emptyset$ — Replay buffer

Repeat (for each episode):

  Initialize *s*

  Repeat (for each step of the episode):

    Choose *a* from *s* using policy derived from Q (e.g. e-greedy)

    Take action *a*, observe *r, s'*

    $D \leftarrow D \cup (s, a, s', r)$ — Add this exp to buffer

    $s \leftarrow s'$

    If mod(*step,trainfreq*) == 0: — Train every *trainfreq* steps

      sample batch *B* from *D*

      $w \leftarrow w - \alpha \nabla_w L(B; w)$ — One step grad descent WRT buffer

Deep Q Learning WITH replay buffers

$$L(B;w) = \frac{1}{2} \sum_{(s,a,s',r) \in B} \left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a) \right)^2$$

– buffers like this are pretty common in DL...

D

Add this exp to buffer

$s \leftarrow s'$

If mod(*step*,*tra* == 0:

sample batch from *D*

$w \leftarrow w - \alpha \nabla_w \boxed{L(B;w)}$

Train every *trainfreq* steps

One step grad descent WRT buffer

Example: 4x4 frozen lake env

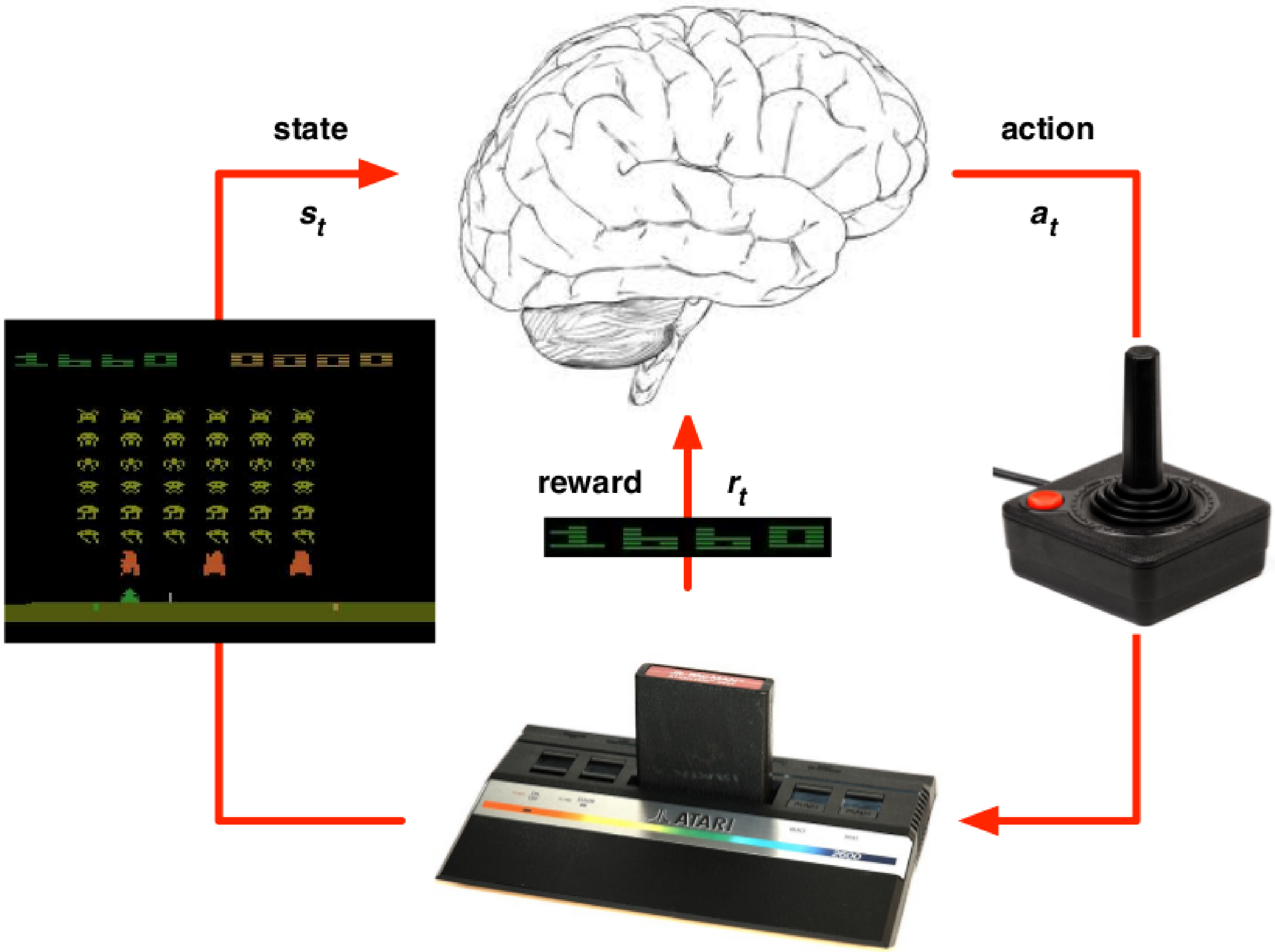
Get to the goal (G)
Don't fall in a hole (H)

```
SFFF
FHFH
FFFH
HFFG
  (Left)
```

```
steps: 11582, episodes: 770, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.0873818397522
steps: 11609, episodes: 771, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.0872020721436
steps: 11652, episodes: 772, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.138998985291
steps: 11672, episodes: 773, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.0649240016937
steps: 11689, episodes: 774, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.0546970367432
steps: 11697, episodes: 775, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.0260739326477
steps: 11731, episodes: 776, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.110991954803
steps: 11773, episodes: 777, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.135339975357
steps: 11798, episodes: 778, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.0810689926147
steps: 11818, episodes: 779, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.0643260478973
steps: 11870, episodes: 780, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.169064044952
steps: 11906, episodes: 781, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.117113113403
steps: 11992, episodes: 782, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.279519796371
steps: 12064, episodes: 783, mean 100 episode reward: 0.6, % time spent exploring: 2, time elapsed: 0.234206199646
steps: 12090, episodes: 784, mean 100 episode reward: 0.7, % time spent exploring: 2, time elapsed: 0.0835938453674
steps: 12137, episodes: 785, mean 100 episode reward: 0.7, % time spent exploring: 2, time elapsed: 0.150979042053
steps: 12185, episodes: 786, mean 100 episode reward: 0.7, % time spent exploring: 2, time elapsed: 0.155304908752
steps: 12245, episodes: 787, mean 100 episode reward: 0.7, % time spent exploring: 2, time elapsed: 0.194122076035
steps: 12277, episodes: 788, mean 100 episode reward: 0.7, % time spent exploring: 2, time elapsed: 0.102608919144
steps: 12293, episodes: 789, mean 100 episode reward: 0.7, % time spent exploring: 2, time elapsed: 0.0520431995392
```

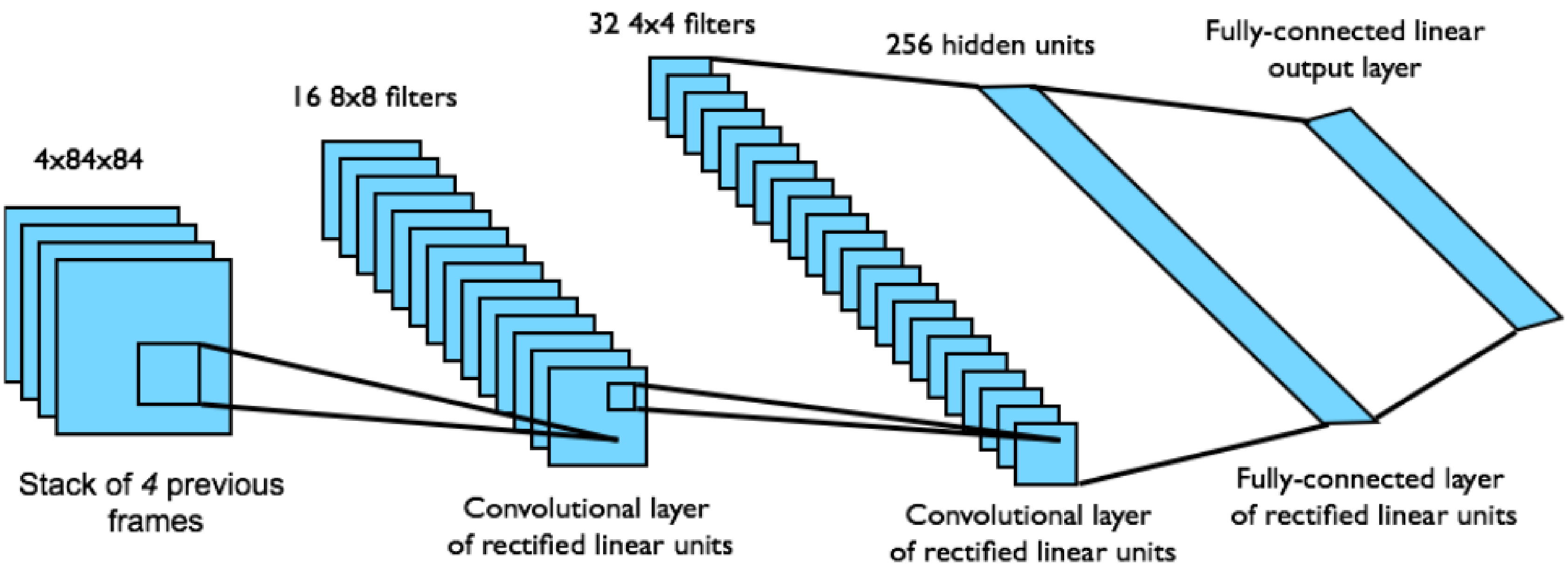
Demo!

Atari Learning Environment



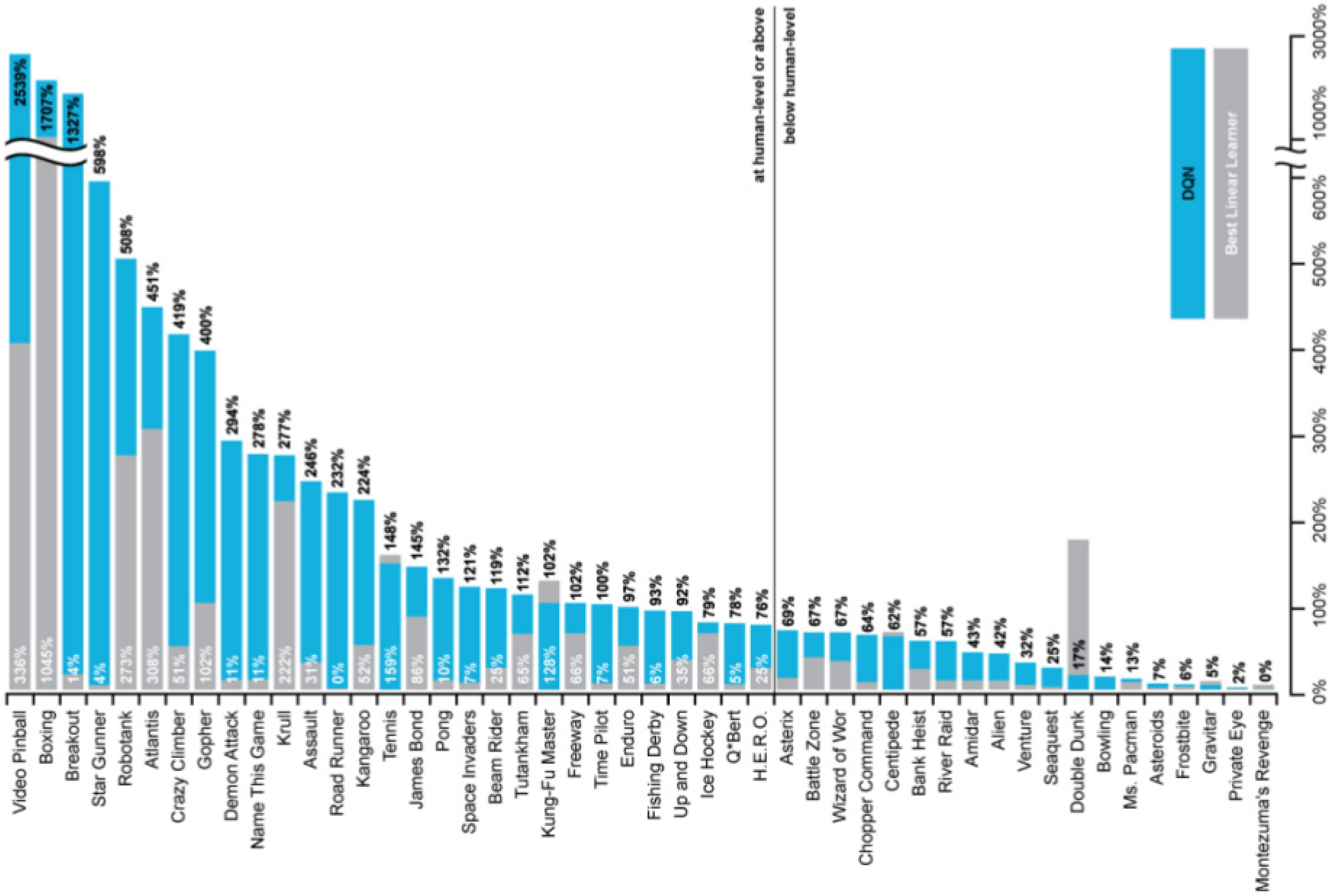
Atari Learning Environment

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

Atari Learning Environment



Additional Deep RL "Tricks"

- Double DQN: Remove upward bias caused by $\max_a Q(s, a, \mathbf{w})$
  - Current Q-network $\mathbf{w}$ is used to select actions
  - Older Q-network $\mathbf{w}^-$ is used to evaluate actions

$$I = \left(r + \gamma Q(s', \underset{a'}{\operatorname{argmax}}\, Q(s', a', \mathbf{w}), \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

Additional Deep RL "Tricks"

- Double DQN: Remove upward bias caused by $\max_a Q(s, a, \mathbf{w})$
  - Current Q-network $\mathbf{w}$ is used to select actions
  - Older Q-network $\mathbf{w}^-$ is used to evaluate actions

$$I = \left(r + \gamma Q(s', \underset{a'}{\operatorname{argmax}}\, Q(s', a', \mathbf{w}), \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

- Prioritised replay: Weight experience according to surprise
  - Store experience in priority queue according to DQN error

$$\left| r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, w) \right|$$

Additional Deep RL "Tricks"

- Double DQN: Remove upward bias caused by $\max_a Q(s, a, \mathbf{w})$
  - Current Q-network $\mathbf{w}$ is used to select actions
  - Older Q-network $\mathbf{w}^-$ is used to evaluate actions

$$I = \left(r + \gamma Q(s', \underset{a'}{\operatorname{argmax}}\, Q(s', a', \mathbf{w}), \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

- Prioritised replay: Weight experience according to surprise
  - Store experience in priority queue according to DQN error

$$\left| r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, w) \right|$$

- Duelling network: Split Q-network into two channels
  - Action-independent value function $V(s, v)$
  - Action-dependent advantage function $A(s, a, \mathbf{w})$

$$Q(s, a) = V(s, v) + A(s, a, \mathbf{w})$$

Additional Deep RL "Tricks"

- Double DQN: Remove upward bias caused by $\max_a Q(s, a, \mathbf{w})$
  - Current Q-network $\mathbf{w}$ is used to select actions
  - Older Q-network $\mathbf{w}^-$ is used to evaluate actions

$$I = \left(r + \gamma Q(s', \operatorname*{argmax}_{a'} Q(s', a', \mathbf{w}), \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

- Prioritised replay: Weight experience according to surprise
  - Store experience in priority queue according to DQN error

$$\left| r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, w) \right|$$

- Duelling network: Split Q-network into two channels
  - Action-independent value function $V(s, v)$
  - Action-dependent advantage function $A(s, a, \mathbf{w})$

$$Q(s, a) = V(s, v) + A(s, a, \mathbf{w})$$

Combined algorithm: 3x mean Atari score DQN w/ replay buffers