Deep RL
Q: I’ve seen the term Markov boundary used before: is that related to a Markov blanket?

A: In a BayesNet, the Markov blanket for X is *any* set S s.t. X is conditionally independent of all other variables when conditioned on S.

The Markov boundary for X is the smallest possible Markov blanket, which happens to be the children, parents and co-parents of X (note this is the definition of a Markov blanket we presented)

Every Markov boundary is a Markov blanket but not vice versa.
STOCHASTIC REWARDS AND VALUE ITERATION
**Q:** What if the rewards are also stochastic?

**A:** No problem. Everything we’ve been doing here still works just fine.

The Q-Learning algorithm doesn’t need to change at all.

Let’s consider how value iteration would look slightly different though...
RL: Components

From the Environment (i.e. the MDP)

- State space, $\mathcal{S}$
- Action space, $\mathcal{A}$
- Reward function, $R(s, a, s')$, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$
- Transition probabilities, $p(s' | s, a)$
  - Deterministic transitions:
    $$p(s' | s, a) = \begin{cases} 1 & \text{if } \delta(s, a) = s' \\ 0 & \text{otherwise} \end{cases}$$

  where $\delta(s, a)$ is a transition function

From the Model

- Policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$
- Value function, $V^\pi : \mathcal{S} \rightarrow \mathbb{R}$
  - Measures the expected total payoff of starting in some state $s$ and executing policy $\pi$
Markov Decision Processes (MDP)

In RL, the source of our data is an MDP:

1. Start in some initial state $s_0 \in S$
2. For time step $t$:
   1. Agent observes state $s_t \in S$
   2. Agent takes action $a_t \in A$ where $a_t = \pi(s_t)$
   3. Agent receives reward $r_t \in \mathbb{R}$ where $r_t = R(s_t, a_t, s_{t+1})$
   4. Agent transitions to state $s_{t+1} \in S$ where $s_{t+1} \sim p(s' \mid s_t, a_t)$
3. Total reward is $\sum_{t=0}^{\infty} \gamma^t r_t$
   - The value $\gamma$ is the “discount factor”, a hyperparameter $0 < \gamma < 1$

- Makes the same Markov assumption we used for HMMs! The next state only depends on the current state and action.
- \textit{Def.}: we \textbf{execute} a policy $\pi$ by taking action $a = \pi(s)$ when in state $s$
Optimal Value Function

For the optimal policy function $\pi^*$ we can compute its **value function** as:

$$V^{\pi^*}(s) = V^*(s) = \mathbb{E}[R(s_0, \pi^*(s_0), s_1) + \gamma R(s_1, \pi^*(s_1), s_2) + \gamma^2 R(s_2, \pi^*(s_2), s_3) \cdots | s_0 = s, \pi^*].$$

This **optimal value function** can be represented recursively as:

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} p(s'|s,a) (R(s,a,s') + \gamma V^*(s')).$$

If $R(s,a,s') = R(s,a)$ (deterministic transition), then we have the form:

$$V^*(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)V^*(s') \right\}.$$
Value Iteration

**Algorithm 1** Value Iteration with Stochastic Rewards

1. **procedure** VALUEITERATION($R(s, a, s')$ reward function, $p(\cdot | s, a)$ transition probabilities)
2. Initialize value function $V(s) = 0$ or randomly
3. while not converged do
4.   for $s \in S$ do
5.     $V(s) = \max_a \sum_{s' \in S} p(s'|s, a)(R(s, a, s') + \gamma V(s'))$
6.   Let $\pi(s) = \arg\max_a \sum_{s' \in S} p(s'|s, a)(R(s, a, s') + \gamma V(s'))$, $\forall s$
7. return $\pi$

This is just fixed point iteration applied to the recursive definition of the optimal value function.
RL: Value Function Example

$$R(s, a) = \begin{cases} 
-2 & \text{if entering state 0 (safety)} \\
3 & \text{if entering state 5 (field goal)} \\
7 & \text{if entering state 6 (touch down)} \\
0 & \text{otherwise} 
\end{cases}$$

$$\gamma = 0.9$$
$R(s, a) = \begin{cases} 
-2 & \text{if entering state 0 (safety)} \\
3 & \text{if entering state 5 (field goal)} \\
7 & \text{if entering state 6 (touch down)} \\
0 & \text{otherwise} 
\end{cases}$

$\gamma = 0.9$
$R(s, a) = \begin{cases} 
-2 & \text{if entering state 0 (safety)} \\
3 & \text{if entering state 5 (field goal)} \\
7 & \text{if entering state 6 (touch down)} \\
0 & \text{otherwise}
\end{cases}$

$\gamma = 0.9$
Example: Stochastic Transitions and Rewards

\(\gamma = 0.9\)

\[ R(s, a, s') = \begin{cases} 
-2 & \text{if entering state 0 (safety)} \\
3 & \text{if entering state 5 (field goal)} \\
7 & \text{if entering state 6 (touch down)} \\
0 & \text{otherwise} 
\end{cases} \]
Example: Stochastic Transitions and Rewards

\[ R(s, a, s') \] represented by \[ \gamma = 0.9 \]

Suppose
- \( p(s6 \mid s4, a) = 0.5 \)
- \( p(s5 \mid s4, a) = 0.5 \)

What is \( V^*(s4) \)?
Example: Stochastic Transitions and Rewards

$R(s, a, s')$ represented by
\[ \gamma = 0.9 \]

Suppose
\[ \cdot p(s6 \mid s4, \ a) = 0.5 \]
\[ \cdot p(s5 \mid s4, \ a) = 0.5 \]

What is $V^*(s4)$?
Q-LEARNING
Learning $Q^*(s, a)$ w/ deterministic transitions

- **Algorithm 1**: Online learning of $Q^*$ (table form)
  - Inputs: discount factor $\gamma$,
    an initial state $s$

  - Initialize $Q(s, a) = 0 \forall s \in \mathcal{S}, a \in \mathcal{A}$
    ($Q$ is a $|\mathcal{S}| \times |\mathcal{A}|$ table or array)

  - While TRUE, do
    - Take a random action $a$
    - Receive some reward $r = R(s, a)$
    - Observe the new state $s' = \delta(s, a)$
    - Update $Q$ and $s$
      $Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$
      $s \leftarrow s'$

Online gathering of training sample $(s, a, r, s')$
Learning $Q^*(s, a)$ w/ deterministic transitions

- Algorithm 2: $\epsilon$-greedy online learning of $Q^*$ (table form)
  - Inputs: discount factor $\gamma$, an initial state $s$, greediness parameter $\epsilon \in [0, 1]$
  - Initialize $Q(s, a) = 0 \ \forall \ s \in S, a \in A$
    (Q is a $|S| \times |A|$ table or array)
  - While TRUE, do
    - With probability $1 - \epsilon$, take the greedy action $a = \text{argmax}_{a'} Q(s, a')$. Otherwise (with probability $\epsilon$), take a random action $a$
    - Receive reward $r = R(s, a)$
    - Observe the new state $s' = \delta(s, a)$
    - Update $Q$ and $s$
      $$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$
      $$s \leftarrow s'$$
Algorithm 3: $\epsilon$-greedy online learning of $Q^*$ (table form)

- **Inputs:** discount factor $\gamma$, an initial state $s$, greediness parameter $\epsilon \in [0, 1]$, learning rate $\alpha \in [0, 1]$ (“mistrust parameter”)
- **Initialize** $Q(s, a) = 0 \ \forall \ s \in \mathcal{S}, a \in \mathcal{A}$ ($Q$ is a $|\mathcal{S}| \times |\mathcal{A}|$ table or array)
- **While** TRUE, **do**
  - With probability $1 - \epsilon$, take the greedy action $a = \text{argmax}_{a' \in \mathcal{A}} Q(s, a')$. Otherwise (with probability $\epsilon$), take a random action $a$
  - Receive reward $r = R(s, a)$
  - Observe the new state $s' \sim p(S' | s, a)$
  - Update $Q$ and $s$
    $$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left( r + \gamma \max_{a'} Q(s', a') \right)$$
    $$s \leftarrow s'$$

Current value

Update w/ deterministic transitions
Algorithm 3: $\epsilon$-greedy online learning of $Q^*$ (table form)

- **Inputs:**
  - discount factor $\gamma$,
  - an initial state $s$,
  - greediness parameter $\epsilon \in [0, 1]$,
  - learning rate $\alpha \in [0, 1]$ (“mistrust parameter”)

- **Initialize** $Q(s, a) = 0 \forall s \in S, a \in A$
  ($Q$ is a $|S| \times |A|$ table or array)

- **While** TRUE, **do**
  - With probability $1 - \epsilon$, take the greedy action $a = \arg\max_{a'} Q(s, a')$. Otherwise (with probability $\epsilon$), take a random action $a$
  - Receive reward $r = R(s, a)$
  - Observe the new state $s' \sim p(S' | s, a)$
  - Update $Q$ and $s$
    $$Q(s, a) \leftarrow Q(s, a) + \alpha \left( r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$
    $$s \leftarrow s'$$

Temporal difference

Current value

Temporal difference target
Learning $Q^*(s, a)$: Example

$\gamma = 0.9$

$R(s, a) = \begin{cases} 
-2 & \text{if entering state 0 (safety)} \\
3 & \text{if entering state 5 (field goal)} \\
7 & \text{if entering state 6 (touch down)} \\
0 & \text{otherwise}
\end{cases}$
Learning $Q^*(s, a)$: Example

$R(s, a)$ represented by $\gamma = 0.9$
Poll Q1: Which set of blue arrows (roughly) corresponds to $Q^*(s, a)$?

$\gamma = 0.9$
Poll Q1: Which set of blue arrows corresponds to $Q^* (s, a)$?

$V^*(s)$ shown in green

$Q^* (s, a) = R(s, a) + \gamma V^*(\delta(s, a))$

C. 

D.
Learning $Q^*(s, a)$: Example

$R(s, a)$ represented by

$\gamma = 0.9$

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
\downarrow & 0 & -2 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\uparrow & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\leftarrow & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Learning $Q^*(s, a)$: Example

$R(s, a)$ represented by →

$\gamma = 0.9$

$Q(3, \rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow, \leftarrow, \uparrow, \downarrow\}} Q(4, a') = 0$

<table>
<thead>
<tr>
<th>$Q(s, a)$</th>
<th>$\rightarrow$</th>
<th>$\leftarrow$</th>
<th>$\uparrow$</th>
<th>$\downarrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Learning $Q^*(s, a)$: Example

$R(s, a)$ represented by →

$\gamma = 0.9$

<table>
<thead>
<tr>
<th>$Q(s, a)$</th>
<th>→</th>
<th>←</th>
<th>↑</th>
<th>↪</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Learning $Q^*(s, a)$: Example

$R(s, a)$ represented by 

$\gamma = 0.9$

$Q(4, \uparrow) \leftarrow 3 + (0.9) \max_{a' \in \{\rightarrow, \leftarrow, \uparrow, \downarrow\}} Q(5, a') = 3$

<table>
<thead>
<tr>
<th>$Q(s, a)$</th>
<th>$\rightarrow$</th>
<th>$\leftarrow$</th>
<th>$\uparrow$</th>
<th>$\downarrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\circ$</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Learning $Q^*(s, a)$: Example

$R(s, a)$ represented by $\gamma = 0.9$

$Q(3, \rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow, \leftarrow, \uparrow, \downarrow\}} Q(4, a') = 2.7$

<table>
<thead>
<tr>
<th>$Q(s, a)$</th>
<th>$\rightarrow$</th>
<th>$\leftarrow$</th>
<th>$\uparrow$</th>
<th>$\downarrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Learning $Q^*(s, a)$: Example

$R(s, a)$ represented by

$\gamma = 0.9$

$Q(3, \rightarrow) \leftarrow 0 + (0.9) \max_{a' \in \{\rightarrow, \leftarrow, \uparrow, \downarrow\}} Q(4, a') = 2.7$

<table>
<thead>
<tr>
<th>$Q(s, a)$</th>
<th>$\rightarrow$</th>
<th>$\leftarrow$</th>
<th>$\uparrow$</th>
<th>$\downarrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2.7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Q-Learning Convergence

Remarks

– Q converges to $Q^*$ with probability 1.0, assuming...
  1. each $<s, a>$ is visited infinitely often
  2. $0 \leq \gamma < 1$
  3. rewards are bounded $|R(s,a)| < \beta$, for all $<s,a>$
  4. initial Q values are finite
  5. Learning rate $\alpha_t$ follows some “schedule” s.t. $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 = 0$, e.g., $\alpha_t = \frac{1}{t+1}$

– Q-Learning is exploration insensitive
  $\Rightarrow$ visiting the states in any order will work assuming point 1 is satisfied

– May take many iterations to converge in practice
Reordering Experiences

**Ex: Easiest Maze Ever!**

\[ y = 0.9 \]

\[ S = \{ A, B, C, D \} \]

\[ A = \{ E, N \} \]

\[ Q(s,a) = 0 \text{ at start} \]

1. Suppose we visit

<table>
<thead>
<tr>
<th>States</th>
<th>Actions</th>
<th>Q-values</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>E</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>E</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>E</td>
<td>100</td>
</tr>
</tbody>
</table>

2. Suppose we visit in reverse order

<table>
<thead>
<tr>
<th>States</th>
<th>Actions</th>
<th>Q-values</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>E</td>
<td>100</td>
</tr>
<tr>
<td>B</td>
<td>E</td>
<td>90</td>
</tr>
<tr>
<td>A</td>
<td>E</td>
<td>81</td>
</tr>
</tbody>
</table>
Q: Do we have to retrain our RL agent every time we change our state space?

A: Yes. But whether your state space changes from one setting to another is determined by your design of the state representation.

Two examples:

– State Space A: \(<x, y>\) position on map
e.g. \(s_t = <74, 152>\)

– State Space B: window of pixel colors centered at current Pac Man location

e.g. \(s_t = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \)
Question:
For the $R(s,a)$ values shown on the arrows below, which are the corresponding $Q^*(s,a)$ values? Assume discount factor = 0.5.

Answer:
DEEP RL FOR GAME OF GO
TD Gammon $\rightarrow$ Alpha Go

Learning to beat the masters at board games

<table>
<thead>
<tr>
<th>THEN</th>
<th>NOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>“…the world’s top computer program for backgammon, TD-GAMMON (Tesauro, 1992, 1995), learned its strategy by playing over one million practice games against itself…”</td>
<td><img src="image" alt="AlphaGo diagram" /></td>
</tr>
</tbody>
</table>

(Mitchell, 1997)
Alpha Go

Game of Go (圍棋)
• 19x19 board
• Players alternately play black/white stones
• Goal is to fully encircle the largest region on the board
• Simple rules, but extremely complex game play

AlphaGo (Black) vs. Lee Sedol (White) - Game 2
Final position (AlphaGo wins in 211 moves)

Source: https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol
Alpha Go

- State space is too large to represent explicitly since 
  # of sequences of moves is $O(b^d)$ 
  - Go: $b=250$ and $d=150$ 
  - Chess: $b=35$ and $d=80$ 
- Key idea: 
  - Define a neural network to approximate the value function 
  - Train by policy gradient

![Diagram of Alpha Go](image)
• Results of a tournament
• From Silver et al. (2016): “a 230 point gap corresponds to a 79% probability of winning”
DEEP Q-LEARNING
Deep Q-Learning

**Question:** What if our state space $S$ is too large to represent with a table?

**Examples:**
- $s_t =$ pixels of a video game
- $s_t =$ continuous values of a sensors in a manufacturing robot
- $s_t =$ sensor output from a self-driving car

**Answer:** Use a parametric function to approximate the table entries

**Key Idea:**
1. Use a neural network $Q(s,a; \theta)$ to approximate $Q^*(s,a)$
2. Learn the parameters $\theta$ via SGD with training examples $< s_t, a_t, r_t, s_{t+1} >$
Deep Q-learning

- How can we handle infinite (or just very large) state/action spaces?
- Just throw a neural network at it
- Use a parametric function $Q(s, a; \Theta)$ to approximate $Q^*(s, a)$
  - Learn the parameters using stochastic gradient descent
  - Training data $(s_t, a_t, r_t, s_t)$ gathered online by the agent/learning algorithm
Deep Q-learning: Model

- Represent states using some feature vector $\bar{s}_t \in \mathbb{R}^M$
  e.g., $\bar{s}_t = [1, 0, 0, ..., 1]^T$
- Define a neural network

Model 1:
- $\bar{s}_t \rightarrow \Theta \rightarrow Q(s_t, a_t; \Theta)$
- $a_t$

Model 2: $\bar{s}_t$ (K = $|\mathcal{A}|$)
- $\bar{s}_t \rightarrow \Theta \rightarrow Q(s_t, a_1; \Theta)$
- $Q(s_t, a_2; \Theta)$
- $\vdots$
- $Q(s_t, a_K; \Theta)$
• Represent states using some feature vector $\vec{s}_t \in \mathbb{R}^M$
eq \mathbb{R}^M$, e.g., $\vec{s}_t = [1, 0, 0, ..., 1]^T$

• Define a neural network a bunch of linear regressors (technically still neural networks...), one for each action (let $K = |\mathcal{A}|$)

$$Q(\vec{s}, a_k; \Theta) = \theta_k^T \vec{s} \quad \text{where} \quad \Theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_K \end{bmatrix} \in \mathbb{R}^{K \times M}$$

• Goal: $K \times M \ll |\mathcal{S}| \rightarrow$ computational tractability!

• Gradients are easy: $\nabla_{\theta_j} Q(\vec{s}, a_k; \Theta) = \begin{cases} \vec{0} & \text{if } j \neq k \\ \vec{s} & \text{if } j = k \end{cases}$
Deep Q-learning: Model

- Represent states using some feature vector $\vec{s}_t \in \mathbb{R}^M$
eq \begin{bmatrix} 1, 0, 0, \ldots, 1 \end{bmatrix}^T$
- Define a neural network a bunch of linear regressors (technically still neural networks...), one for each action (let $K = |\mathcal{A}|$)

$$Q(\vec{s}, a_k; \Theta) = \theta_k^T \vec{s} \quad \text{where} \quad \Theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_K \end{bmatrix} \in \mathbb{R}^{K \times M}$$

- Goal: $K \times M \ll |S| \rightarrow \text{computational tractability!}$

- Gradients are easy: $\nabla_\Theta Q(\vec{s}, a_k; \Theta) =$ \[ \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \vec{s} \end{bmatrix} \quad \text{Row } k \]
Deep Q-learning: Loss Function

1. $S$ too big to compute this sum

2. Don’t know $Q^*$

   - “True” loss
     \[
     \ell(\Theta) = \sum_{s \in S} \sum_{a \in A} (Q^*(s, a) - Q(s, a; \Theta))^2
     \]

   - Use stochastic gradient descent: just consider one state-action pair in each iteration

   - Use temporal difference learning:
     
     - Given current parameters $\Theta^{(t)}$ the (temporal difference) target is
       \[
       Q^*(s, a) \approx r + \gamma \max_{a'} Q(s', a'; \Theta^{(t)}) \equiv y
       \]

     - Set the parameters in the next iteration $\Theta^{(t+1)}$ such that $Q(s, a; \Theta^{(t+1)}) \approx y$

     \[
     \ell(\Theta^{(t)}, \Theta^{(t+1)}) = (y - Q(s, a; \Theta^{(t+1)}))^2
     \]
Deep Q-learning

- Algorithm 4: Online learning of $Q^*$ (parametric form)
  - Inputs: discount factor $\gamma$, an initial state $s_0$, learning rate $\alpha$
  - Initialize parameters $\Theta^{(0)}$
  - For $t = 0, 1, 2, \ldots$
    - Gather training sample $(s_t, a_t, r_t, s_{t+1})$
    - Update $\Theta^{(t)}$ by taking a step opposite the gradient
      - $\Theta^{(const)} \leftarrow \Theta^{(t)}$
      - $\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \alpha \nabla_{\Theta^{(t)}} \ell(\Theta^{(const)}, \Theta^{(t)})$

where

$$\nabla_{\Theta} \ell(\Theta^{(const)}, \Theta^{(t)}) = 2 \left( y - Q(s, a; \Theta^{(t)}) \right) \nabla_{\Theta^{(t)}} Q(s, a; \Theta^{(t)})$$

$$= 2 \left( r + \gamma \max_{a'} Q(s', a'; \Theta^{(const)}) - Q(s, a; \Theta^{(t)}) \right) \nabla_{\Theta^{(t)}} Q(s, a; \Theta^{(t)})$$
Experience Replay

• **Problems** with online updates for Deep Q-learning:
  – not i.i.d. as SGD would assume
  – quickly forget rare experiences that might later be useful to learn from

• **Uniform Experience Replay** (Lin, 1992):
  – Keep a *replay memory* \( D = \{e_1, e_2, \ldots, e_N\} \) of \( N \) most recent experiences \( e_t = \langle s_t, a_t, r_t, s_{t+1} \rangle \)
  – Alternate two steps:
    1. Repeat \( T \) times: randomly sample \( e_i \) from \( D \) and apply a Q-Learning update to \( e_i \)
    2. Agent selects an action using epsilon greedy policy to receive new experience that is added to \( D \)

• **Prioritized Experience Replay** (Schaul et al, 2016)
  – similar to Uniform ER, but sample so as to prioritize experiences with high error
DEEP RL FOR ATARI GAMES
Playing Atari with Deep RL

- Setup: RL system observes the pixels on the screen.
- It receives rewards as the game score.
- Actions decide how to move the joystick / buttons.

Figures from David Silver (Intro RL lecture)
Playing Atari games with Deep RL

Source: https://www.youtube.com/watch?v=V1eYniJ0Rnk&t=2s&ab_channel=TwoMinutePapers
Playing Atari games with Deep RL

Figure 1: Screen shots from five Atari 2600 Games: (Left-to-right) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

Table 1: The upper table compares average total reward for various learning methods by running an $\epsilon$-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an $\epsilon$-greedy policy with $\epsilon = 0.05$. 

<table>
<thead>
<tr>
<th></th>
<th>B. Rider</th>
<th>Breakout</th>
<th>Enduro</th>
<th>Pong</th>
<th>Q*bert</th>
<th>Seaquest</th>
<th>S. Invaders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>354</td>
<td>1.2</td>
<td>0</td>
<td>−20.4</td>
<td>157</td>
<td>110</td>
<td>179</td>
</tr>
<tr>
<td>Contingency [4]</td>
<td>1743</td>
<td>6</td>
<td>159</td>
<td>−17</td>
<td>960</td>
<td>723</td>
<td>268</td>
</tr>
<tr>
<td>DQN</td>
<td>4092</td>
<td>168</td>
<td>470</td>
<td>20</td>
<td>1952</td>
<td>1705</td>
<td>581</td>
</tr>
<tr>
<td>Human</td>
<td>7456</td>
<td>31</td>
<td>368</td>
<td>−3</td>
<td>18900</td>
<td>28010</td>
<td>3690</td>
</tr>
<tr>
<td>HNeat Best [8]</td>
<td>3616</td>
<td>52</td>
<td>106</td>
<td>19</td>
<td>1800</td>
<td>920</td>
<td>1720</td>
</tr>
<tr>
<td>HNeat Pixel [8]</td>
<td>1332</td>
<td>4</td>
<td>91</td>
<td>−16</td>
<td>1325</td>
<td>800</td>
<td>1145</td>
</tr>
<tr>
<td>DQN Best</td>
<td>5184</td>
<td>225</td>
<td>661</td>
<td>21</td>
<td>4500</td>
<td>1740</td>
<td>1075</td>
</tr>
</tbody>
</table>
Learning Objectives

Reinforcement Learning: Q-Learning

You should be able to...

1. Apply Q-Learning to a real-world environment
2. Implement Q-learning
3. Identify the conditions under which the Q-learning algorithm will converge to the true value function
4. Adapt Q-learning to Deep Q-learning by employing a neural network approximation to the Q function
5. Describe the connection between Deep Q-Learning and regression
BIG PICTURE
**Learning Paradigms:**
What data is available and when? What form of prediction?
- supervised learning
- unsupervised learning
- semi-supervised learning
- reinforcement learning
- active learning
- imitation learning
- domain adaptation
- online learning
- density estimation
- recommender systems
- feature learning
- manifold learning
- dimensionality reduction
- ensemble learning
- distant supervision
- hyperparameter optimization

**Problem Formulation:**
What is the structure of our output prediction?
- boolean: Binary Classification
- categorical: Multiclass Classification
- ordinal: Ordinal Classification
- real: Regression
- ordering: Ranking
- multiple discrete: Structured Prediction
- multiple continuous: (e.g. dynamical systems)
- both discrete & cont.: (e.g. mixed graphical models)

**Facets of Building ML Systems:**
How to build systems that are robust, efficient, adaptive, effective?
1. Data prep
2. Model selection
3. Training (optimization / search)
4. Hyperparameter tuning on validation data
5. (Blind) Assessment on test data

**Theoretical Foundations:**
What principles guide learning?
- probabilistic
- information theoretic
- evolutionary search
- ML as optimization

**Big Ideas in ML:**
Which are the ideas driving development of the field?
- inductive bias
- generalization / overfitting
- bias-variance decomposition
- generative vs. discriminative
- deep nets, graphical models
- PAC learning
- distant rewards

**Application Areas:**
Key challenges?
- NLP, Speech, Computer Vision, Robotics, Medicine, Search
# Learning Paradigms

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised</td>
<td>( \mathcal{D} = { \mathbf{x}^{(i)}, y^{(i)} }_{i=1}^{N} \quad \mathbf{x} \sim p^<em>(\cdot) \text{ and } y = c^</em>(\cdot) )</td>
</tr>
<tr>
<td>( \rightarrow ) Regression</td>
<td>( y^{(i)} \in \mathbb{R} )</td>
</tr>
<tr>
<td>( \rightarrow ) Classification</td>
<td>( y^{(i)} \in {1, \ldots, K} )</td>
</tr>
<tr>
<td>( \rightarrow ) Binary classification</td>
<td>( y^{(i)} \in {+1, -1} )</td>
</tr>
<tr>
<td>( \rightarrow ) Structured Prediction</td>
<td>( y^{(i)} ) is a vector</td>
</tr>
<tr>
<td>Unsupervised</td>
<td>( \mathcal{D} = { \mathbf{x}^{(i)} }_{i=1}^{N} \quad \mathbf{x} \sim p^*(\cdot) )</td>
</tr>
<tr>
<td>Semi-supervised</td>
<td>( \mathcal{D} = { \mathbf{x}^{(i)}, y^{(i)} }<em>{i=1}^{N_1} \cup { \mathbf{x}^{(j)} }</em>{j=1}^{N_2} )</td>
</tr>
<tr>
<td>Online</td>
<td>( \mathcal{D} = { (\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), (\mathbf{x}^{(3)}, y^{(3)}), \ldots } )</td>
</tr>
<tr>
<td>Active Learning</td>
<td>( \mathcal{D} = { \mathbf{x}^{(i)} }_{i=1}^{N} ) and can query ( y^{(i)} = c^*(\cdot) ) at a cost</td>
</tr>
<tr>
<td>Imitation Learning</td>
<td>( \mathcal{D} = { (s^{(1)}, a^{(1)}), (s^{(2)}, a^{(2)}), \ldots } )</td>
</tr>
<tr>
<td>Reinforcement Learning</td>
<td>( \mathcal{D} = { (s^{(1)}, a^{(1)}, r^{(1)}), (s^{(2)}, a^{(2)}, r^{(2)}), \ldots } )</td>
</tr>
</tbody>
</table>