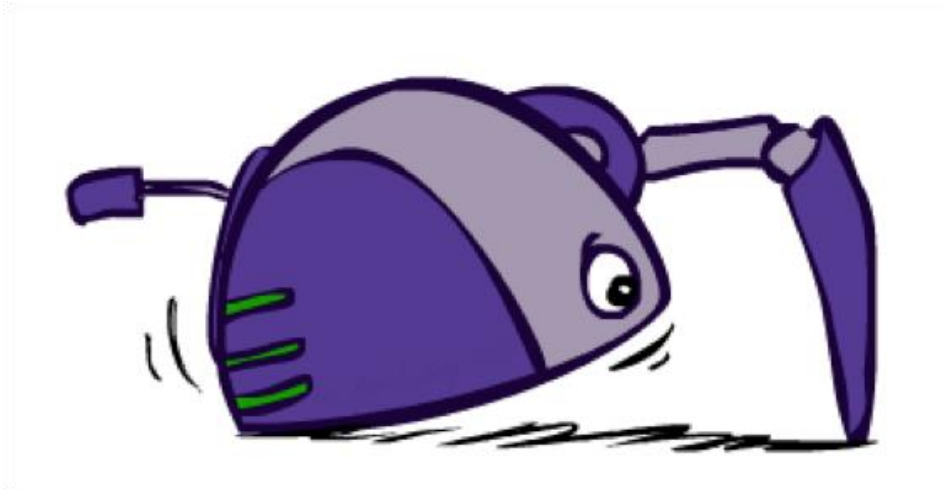
Reinforcement Learning

We still assume an MDP:

- A set of states $s \in S$
- A set of actions (per state) A
- A model T(s,a,s')
- A reward function R(s,a,s')

Still looking for a policy $\pi(s)$



New twist: don't know T or R, so must try out actions

Big idea: Compute all averages over T using sample outcomes

MDP/RL Notation

Standard expectimax: $V(s) = \max_a \sum_{s'} P(s'|s,a)V(s')$

Bellman equations: $V(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$

Value iteration: $V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')]\,, \qquad \forall\, s$

Q-iteration: $Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')]\,, \quad \forall\, s,a$

Policy extraction: $\pi_V(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]\,, \qquad \forall\, s$

Policy evaluation: $V_{k+1}^{\pi}(s) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')]\,, \qquad \forall\, s$

Policy improvement: $\pi_{new}(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi_{old}}(s')]\,, \qquad \forall\, s$

TD (value) learning: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha\,[r + \gamma\, V^{\pi}(s') - V^{\pi}(s)]$

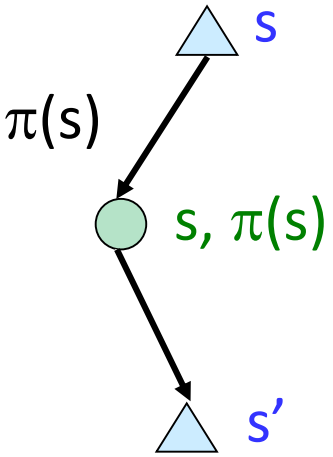
Q-learning: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

Temporal Difference (Value) Learning

Task: Given policy π, learn state value V^{π}

Learn from every experience

- Update $V^{\pi}(s)$ each time we experience a transition (s, a, s', r)
- Likely outcomes s' will contribute updates more often
- Move values toward latest sample (running average)



Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$

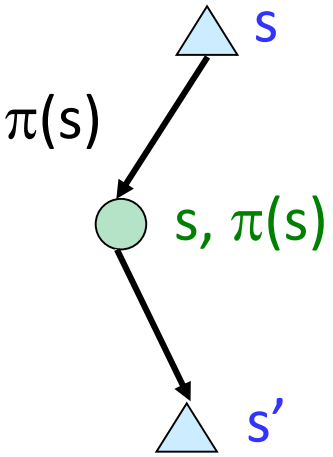
Equivalent to: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

Temporal Difference (Value) Learning

Task: Given policy π, learn state value V^{π}

Learn from every experience

- Update $V^{\pi}(s)$ each time we experience a transition (s, a, s', r)
- Likely outcomes s' will contribute updates more often
- Move values toward latest sample (running average)



Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$

Equivalent to: $\boxed{V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))}$

Equivalent to: $V^{\pi}(s) \leftarrow V^{\pi}(s) - \alpha \nabla Error$ $\qquad Error = \frac{1}{2}\left(sample - V^{\pi}(s)\right)^2$

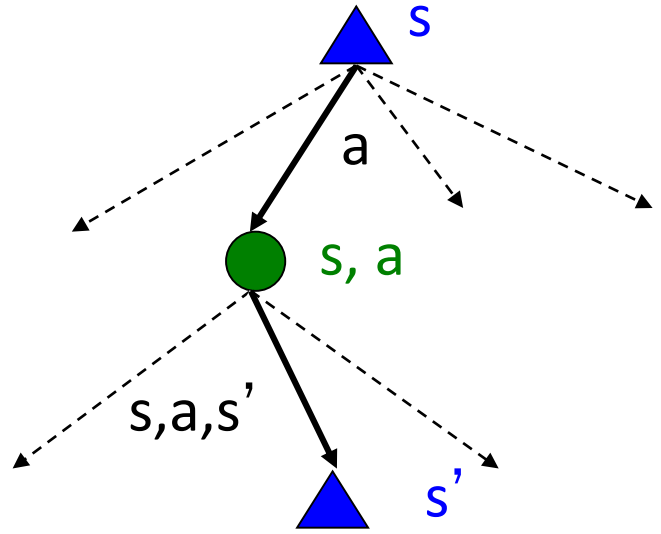
Problems with TD Value Learning

TD value leaning is a model-free way to do policy evaluation, mimicking Bellman updates with running sample averages

However, if we want to turn values into an improved policy, we're sunk:

$$\pi^{new}(s) = \operatorname*{argmax}_a Q^{\pi_{old}}(s,a)$$

$$= \operatorname*{argmax}_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_{old}}(s')]$$



Again, we don't know R and T!

Solution: Directly learn Q-values, not state values

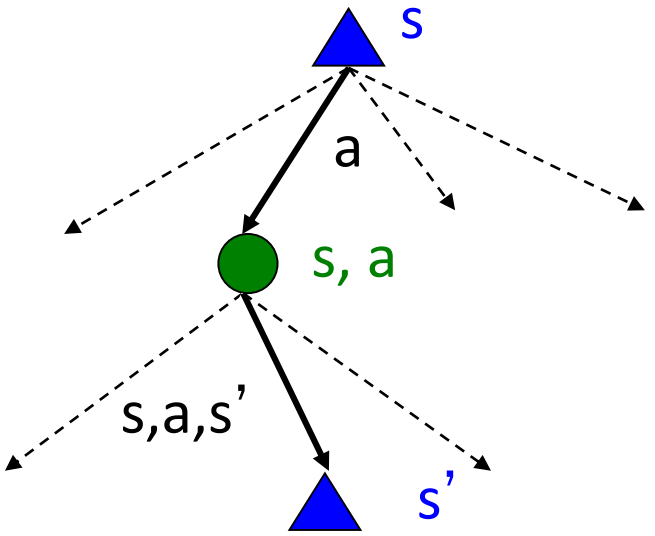
In fact, can directly learn true/optimal Q-values in a model-free way (Q-Learning)

Keep in mind that our ultimate goal is to find optimal policy!

Extending TD Learning to Q-Value

Task: Given policy π, learn state value V^{π}

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha[R(s,a,s') + \gamma V^{\pi}(s') - V^{\pi}(s)]$$
where $a = \pi(s)$



Task: Given policy π, learn Q-state value Q^{π}

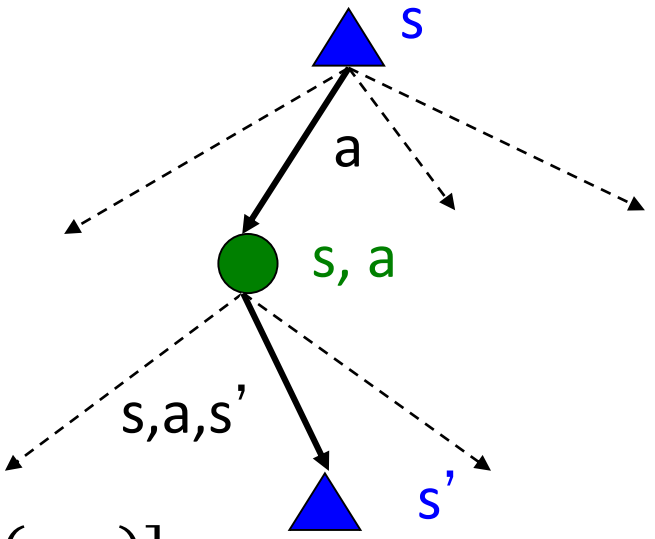
Task: Directly learn true/optimal Q-state value Q

Q-Learning. No given policy π.

Extending TD Learning to Q-Value

Task: Given policy π, learn state value V^π

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha[R(s,a,s') + \gamma V^\pi(s') - V^\pi(s)]$$
where $a = \pi(s)$



Task: Given policy π, learn Q-state value Q^π

$$Q^\pi(s,a) \leftarrow Q^\pi(s,a) + \alpha[R(s,a,s') + \gamma Q^\pi(s',\pi(s')) - Q^\pi(s,a)]$$
where $a = \pi(s)$

Task: Directly learn true/optimal Q-state value Q

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

Q-Learning. No given policy π.

Detour: Q-Value Iteration

Value iteration: find successive (depth-limited) values

- Start with $V_0(s) = 0$, which we know is right
- Given V_k, calculate the depth k+1 values for all states:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

But Q-values are more useful, so compute them instead

- Start with $Q_0(s,a) = 0$, which we know is right
- Given Q_k, calculate the depth k+1 q-values for all q-states:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

Q-Learning

We'd like to do Q-value updates to each Q-state:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- But can't compute this update without knowing T, R

Instead, compute average as we go

- Receive a sample transition (s,a,r,s')
- This sample suggests $Q(s,a) \approx r + \gamma \max_{a'} Q(s',a')$
- But we want to consider our previous value of $Q(s,a)$ (Why?)
- So keep a running average

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

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

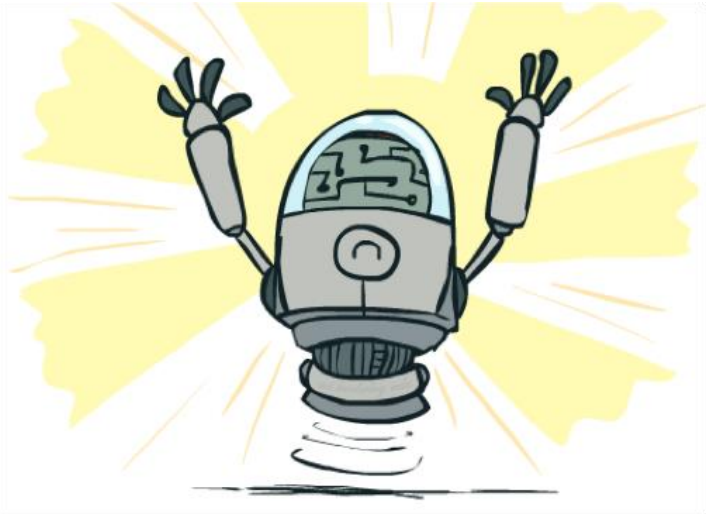
$$Q(s,a) \leftarrow Q(s,a) - \alpha \nabla Error \qquad Error = \frac{1}{2}\left(sample - Q(s,a)\right)^2$$

Q-Learning Properties

Amazing result: Q-learning converges to the Q-value of the optimal policy -- even if you're acting suboptimally!

This is called off-policy learning: you learn the value of the optimal policy while your behavior policy (how you act) is a different policy

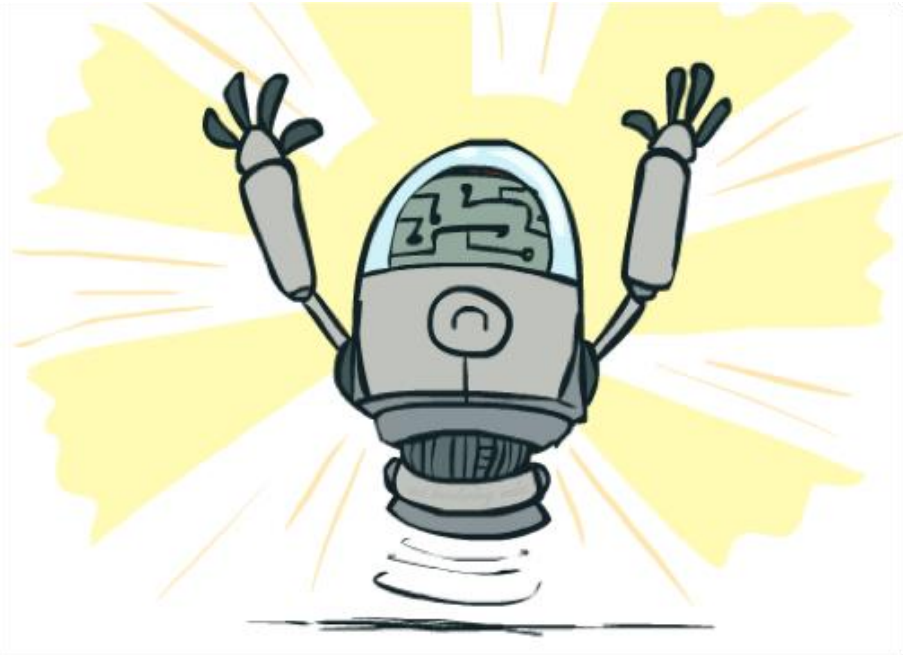
In contrast, on-policy learning (e.g., TD value learning) estimates the value of a policy while acting according to it



Q-Learning Properties

Caveats:

- You have to explore enough
- You have to eventually make the learning rate small enough
- … but not decrease it too quickly
- Basically, in the limit, it doesn't matter how you select actions (!)



The Story So Far: MDPs and RL

Known MDP: Offline Solution

Goal	Technique
Compute V*, Q*, π*	Value / policy iteration
Evaluate a fixed policy π	Policy evaluation

Unknown MDP: Model-Based

Goal	Technique
Compute V*, Q*, π*	VI/PI on approx. MDP
Evaluate a fixed policy π	PE on approx. MDP

Unknown MDP: Model-Free

Goal	Technique
Compute V*, Q*, π*	Q-learning
Evaluate a fixed policy π	TD/Value Learning

Linear Value Functions

Using a feature representation, we can write a Q-value function (or state value function) to approximate the Q-value (or state value) for any state using a few weights:

- $V_{\boldsymbol{w}}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
- $Q_{\boldsymbol{w}}(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$

Advantage: our experience is summed up in a few powerful numbers

Disadvantage: states may share features but actually be very different in value!

Approximate Q-Learning: Q-Learning with Q-value function (a.k.a. Q-function)

Updating a linear value function

Original Q-learning: Update Q values directly (stored in a table)

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

Latest sample — Previous estimate

Difference

Can be viewed as trying to reduce prediction error at s, a:

$$Q(s,a) \leftarrow Q(s,a) - \alpha \nabla Error \qquad Error = \frac{1}{2}\left(sample - Q(s,a)\right)^2$$

Approximate Q-Learning with Linear Q-Value Function:

$$Q_w(s,a) = w_1 f_1(s,a) + \ldots + w_n f_n(s,a)$$

Update weights to reduce prediction error at s, a:

$$w_i \leftarrow w_i - \alpha \frac{\partial Error(w_1, w_2, \ldots, w_n)}{\partial w_i} \qquad Error(w) = \frac{1}{2}\left(sample - Q_w(s,a)\right)^2$$

Updating a linear value function

$$Q_w(s,a) = w_1 f_1(s,a) + \ldots + w_n f_n(s,a)$$

$$w_i \leftarrow w_i - \alpha \frac{\partial Error(w_1, w_2, \ldots, w_n)}{\partial w_i} \qquad Error(w) = \frac{1}{2}\left(sample - Q_w(s,a)\right)^2$$

$$\frac{\partial Error(w)}{\partial w_i} = (Q_w(s,a) - sample)\frac{\partial Q_w(s,a)}{\partial w_i}$$
$$= (Q_w(s,a) - sample) f_i(s,a)$$

Final Update Rule for Approximate Q-Learning with Linear Q-Value Function:

$$\boxed{w_i} \leftarrow w_i + \alpha\left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a)\right)\boxed{f_i(s,a)}$$

Original Q-Learning Update Rule:

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

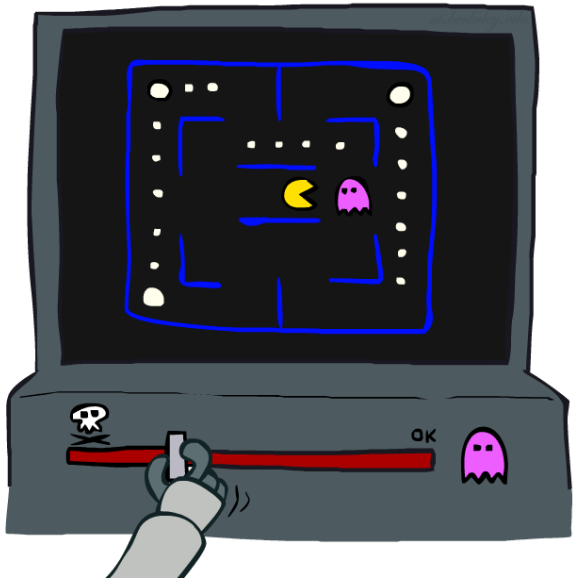
Approximate Q-Learning

Update Rule for Approximate Q-Learning with Linear Q-Value Function:

$$w_i \leftarrow w_i + \alpha \left(r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a) \right) f_i(s, a)$$

Qualitative justification:

- Pleasant surprise: increase weights on +valued features, decrease on – ones
  - As a result, Q_w increased for states with the same (similar) features too. Will now prefer all states with that state's features.
- Unpleasant surprise: decrease weights on +valued features, increase on – ones
  - Disprefer all states with that state's features



What if non-linear value function?

Update Rule for Q-Learning:

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

Update Rule for Approximate Q-Learning with Linear Q-Value Function:

$$w_i \leftarrow w_i + \alpha \left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a) \right) f_i(s,a)$$

Update Rule for Approximate Q-Learning with differentiable Q-function $Q_w(s,a)$:

What if non-linear value function?

Update Rule for Q-Learning:

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

Update Rule for Approximate Q-Learning with Linear Q-Value Function:

$$w_i \leftarrow w_i + \alpha \left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a)\right) f_i(s,a)$$

Update Rule for Approximate Q-Learning with differentiable Q-function $Q_w(s,a)$:

$$w_i \leftarrow w_i + \alpha \left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a)\right) \frac{\partial Q_w(s,a)}{\partial w_i}$$

If $Q_{\boldsymbol{w}}(s,a) = w_1 f_1(s,a) + \ldots + w_n f_n(s,a)$

$$\frac{\partial Q_w(s,a)}{\partial w_i} = f_1(s,a)$$

What if non-linear value function?

Update Rule for Approximate Q-Learning with Q-function $Q_w(s,a)$:

$$w_i \leftarrow w_i + \alpha \left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a) \right) \frac{\partial Q_w(s,a)}{\partial w_i}$$

Why?

$$w_i \leftarrow w_i - \alpha \frac{\partial Error(w_1, w_2, \dots, w_n)}{\partial w_i} \qquad Error(w) = \frac{1}{2} \left(sample - Q_w(s,a) \right)^2$$

$$\frac{\partial Error(w)}{\partial w_i} = (Q_w(s,a) - sample) \frac{\partial Q_w(s,a)}{\partial w_i}$$

$$w_i - \alpha \frac{\partial Error(w_1, w_2, \dots, w_n)}{\partial w_i} = w_i + \alpha \left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a) \right) \frac{\partial Q_w(s,a)}{\partial w_i}$$

What if non-linear value function?

Update Rule for Approximate Q-Learning with Q-function $Q_w(s,a)$:

$$w_i \leftarrow w_i + \alpha \left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a) \right) \frac{\partial Q_w(s,a)}{\partial w_i}$$

Example: $Q_w(s,a) = \exp(w_1 f_1(s,a) + w_2 f_2(s,a) + \cdots + w_n f_n(s,a))$

$$\frac{\partial Q_w(s,a)}{\partial w_i} = \exp\left(w_1 f_1(s,a) + w_2 f_2(s,a) + \cdots + w_n f_n(s,a)\right) f_i(s,a)$$
$$= Q_w(s,a) f_i(s,a)$$

Update Rule:

$$w_i \leftarrow w_i + \alpha \left(r + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a) \right) Q_w(s,a) f_i(s,a)$$