

# reinforcement learning

# pavlov's dog

nervous system



digestion



# robot flipping pancakes

## **Robot Motor Skill Coordination with EM-based Reinforcement Learning**

**Petar Kormushev, Sylvain Calinon,  
and Darwin G. Caldwell**

**Italian Institute of Technology**

<http://cs.stanford.edu/groups/littledog/>



# another example

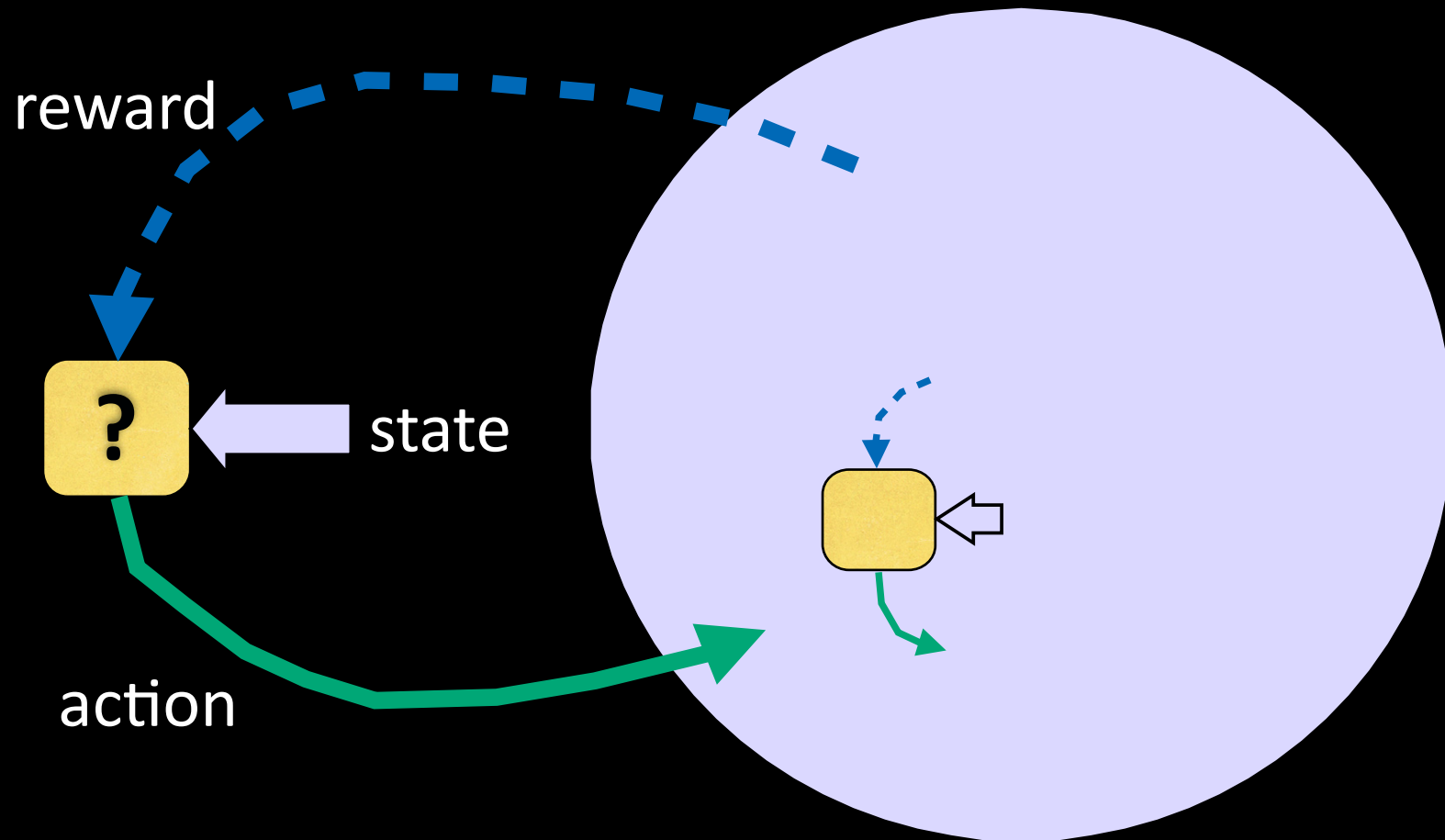
- a child learning to walk:
  - tries out many different strategies
  - some do not work (**falling**), some seem to work (**staying up longer and longer**)
  - the **ones that do not work are discarded**
  - the **ones that work are tried again and again** until perfected or replaced by better strategies

# hovering... inverted!

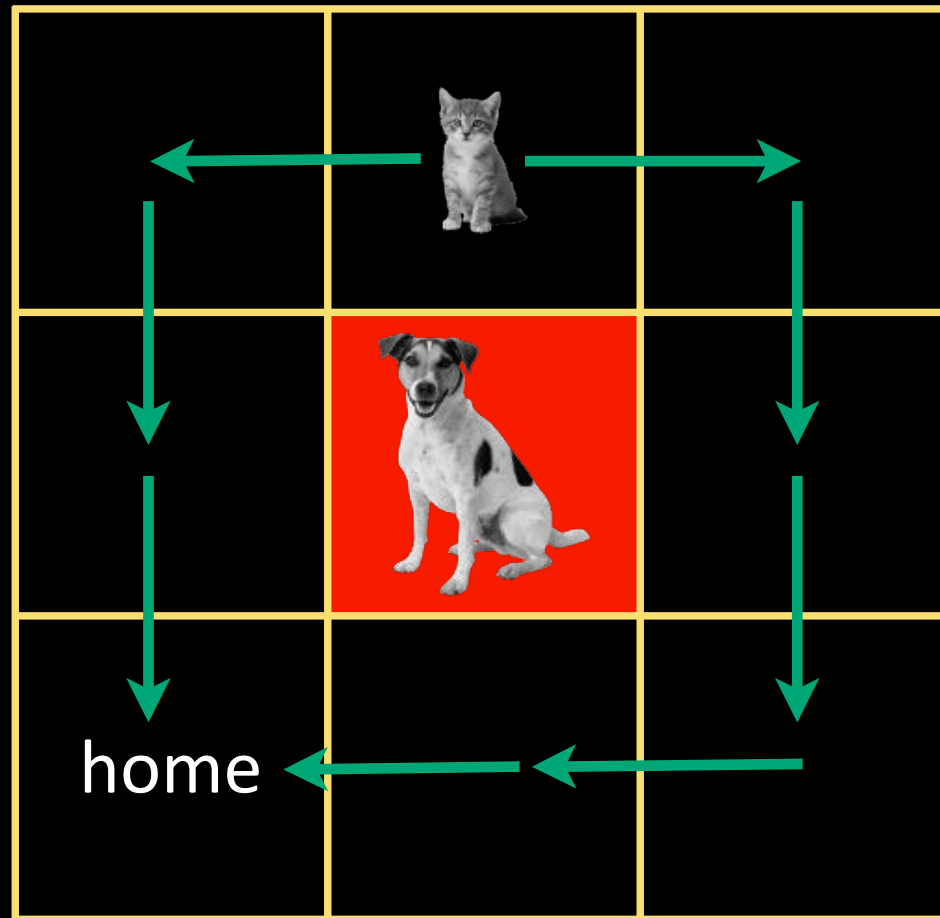


**Inverted autonomous helicopter flight via reinforcement learning**, Andrew Y. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger and Eric Liang. In *International Symposium on Experimental Robotics*, 2004.  
URL: <http://heli.stanford.edu/>

# the problem



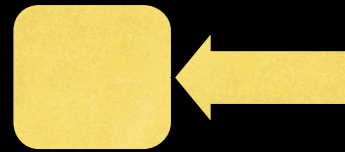
# toy problem





# state and action spaces

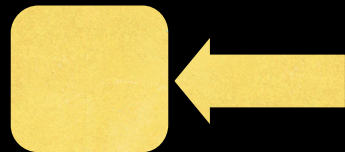
- size of these spaces can be **quite large**
- specifying the spaces is crucial in designing a good learning agent



5 integer values between  
1 and 100: {22,44,12,67,9}

size of state space =  $100 \times 100 \times 100 \times 100 \times 100$

can quantise state space differently

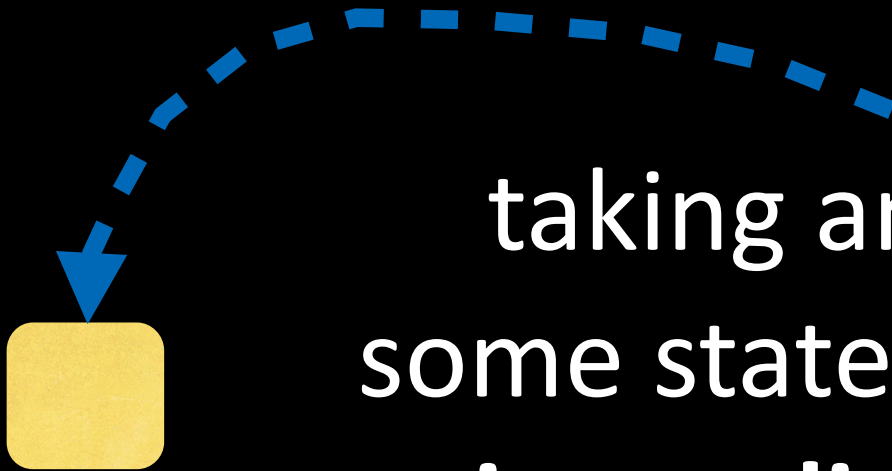


5 values belonging  
to 2 classes: {1, 2, 1, 2, 1}

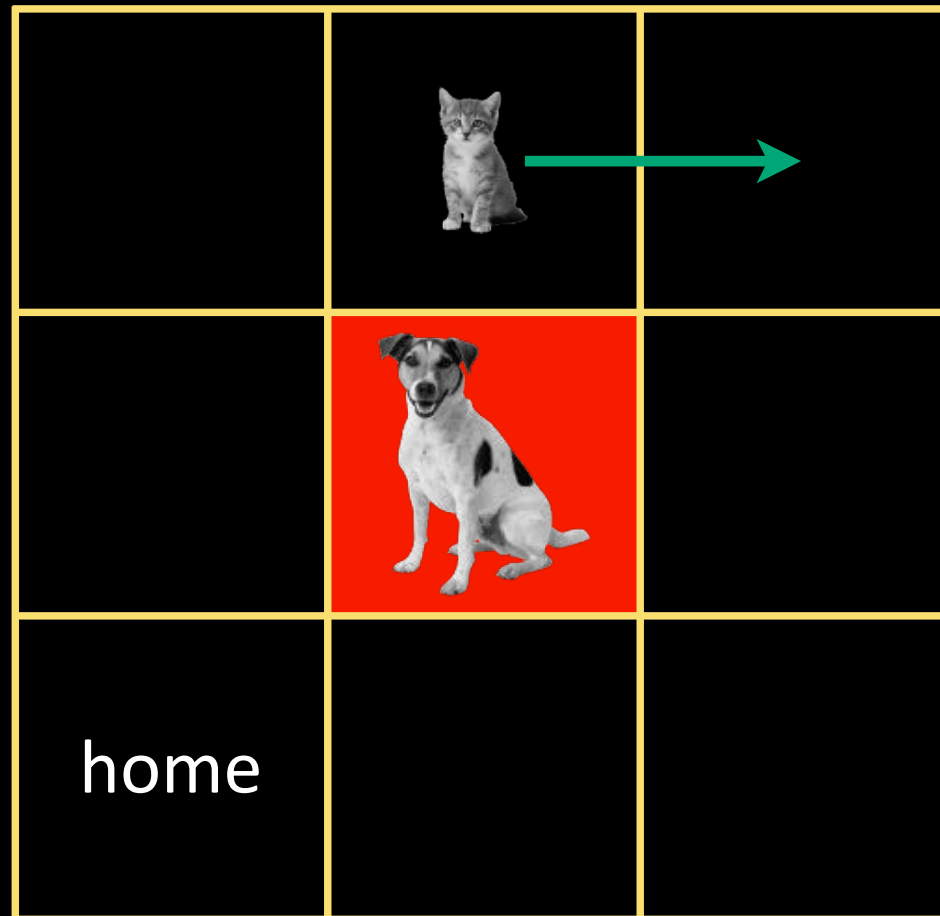
size of state space =  $2 \times 2 \times 2 \times 2 \times 2$

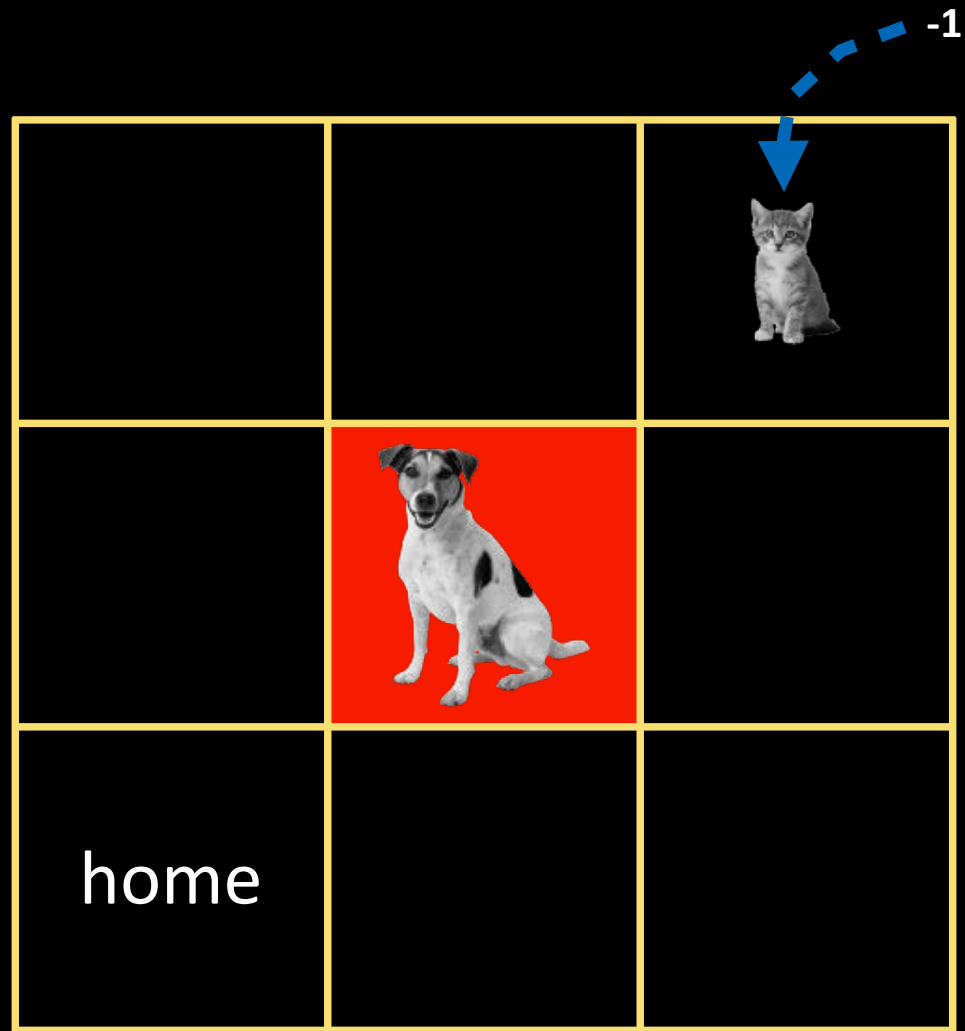
in the toy problem? 9

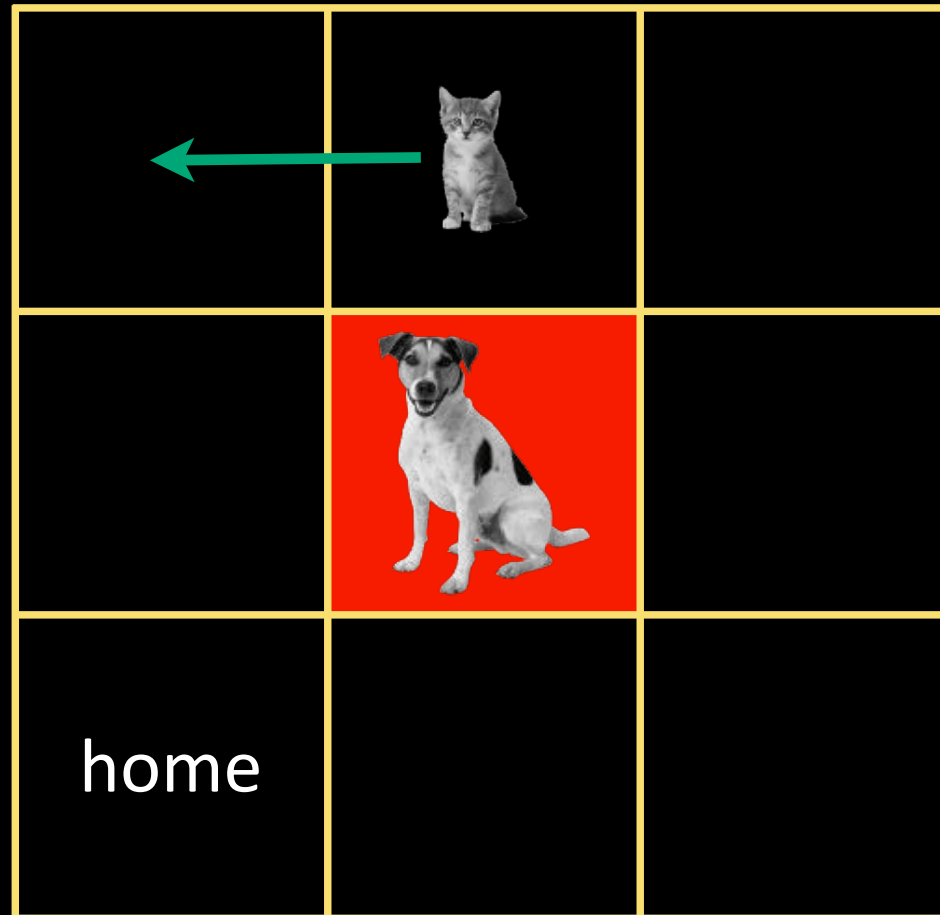
reward

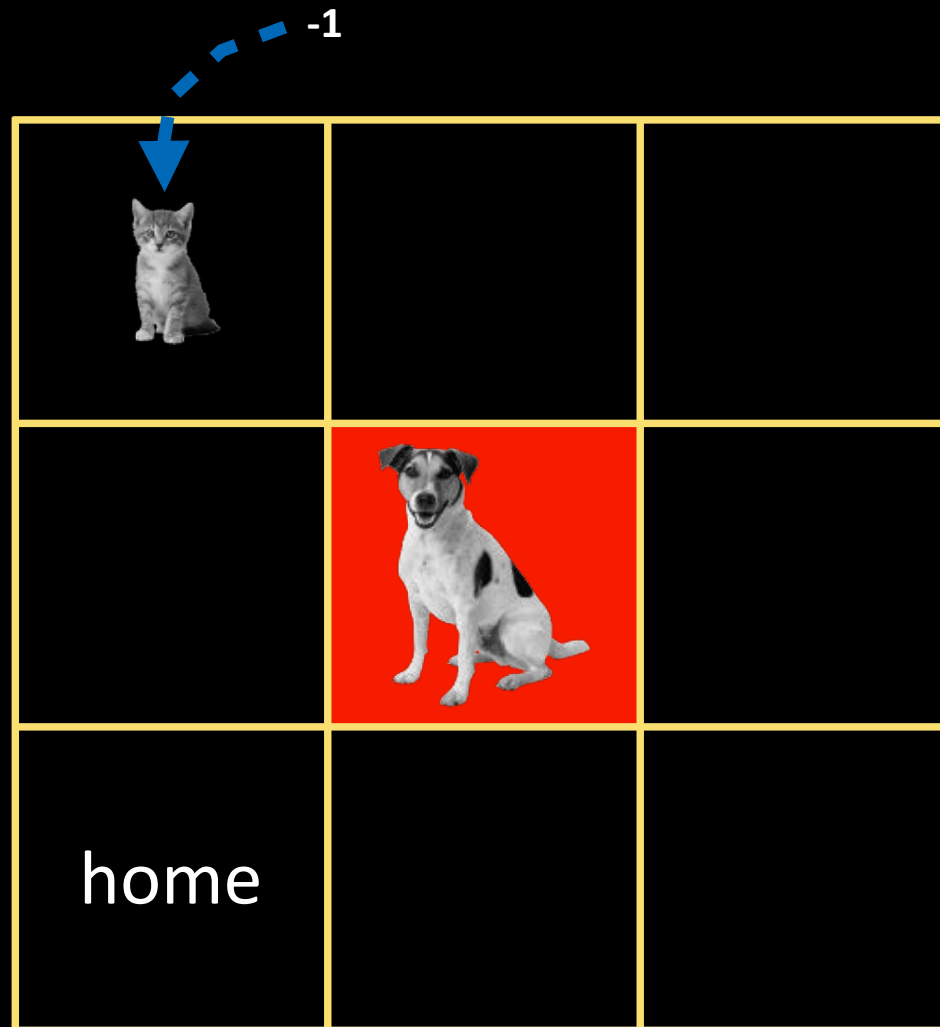


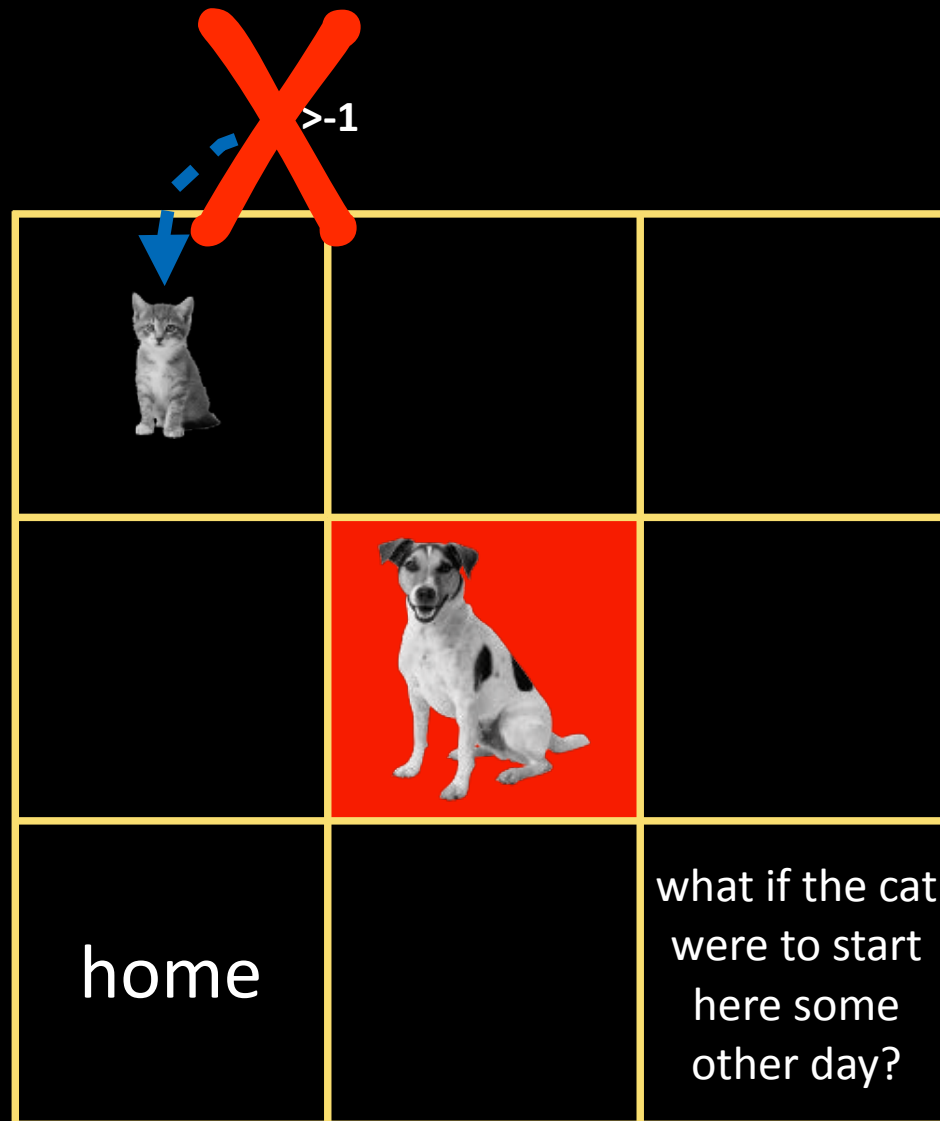
taking an action in  
some state results in an  
**immediate reward**  
(can be negative)











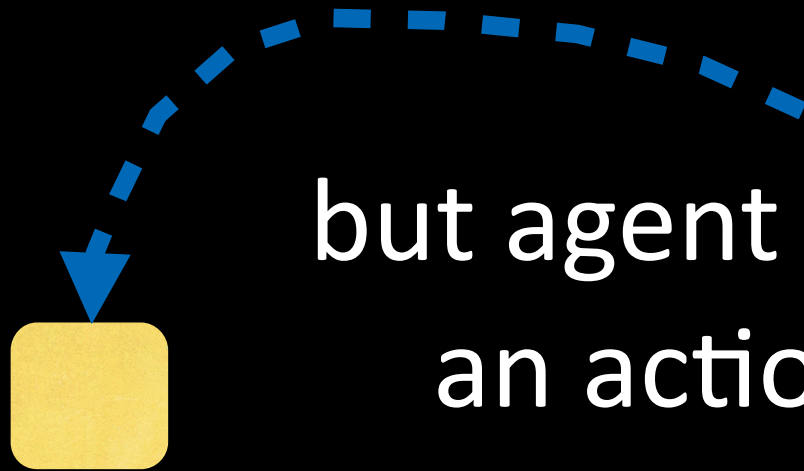


reward system should tell  
the agent:

**what to achieve**

rather than how to achieve

reward?



but agent has to choose  
an action based on  
**expected “long term”**  
reward (cumulative  
reward in the long run)

**expected “long term”  
reward (cumulative  
reward in the long run)**



task

episodic

continual

(there is an **end**)

(there is no **end**)

**episodic**

(there is an **end**)

agent taking **finite (say 5) steps** till the end...

should act based on the  
**average of the following**

$$R_0 = r_1 + r_2 + r_3 + r_4 + r_5$$

**continual**

(there is no **end**)

agent can continue acting for **infinite steps**  
**in time...**

should **discount** future rewards and act based on  
the **average of the following**

$$R_0 = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \gamma^4 r_5 + \dots$$

# discount

future reward is probably more  
uncertain than immediate reward

shortsighted?

$\gamma=0$

$$0 \leq \gamma \leq 1$$

farsighted?

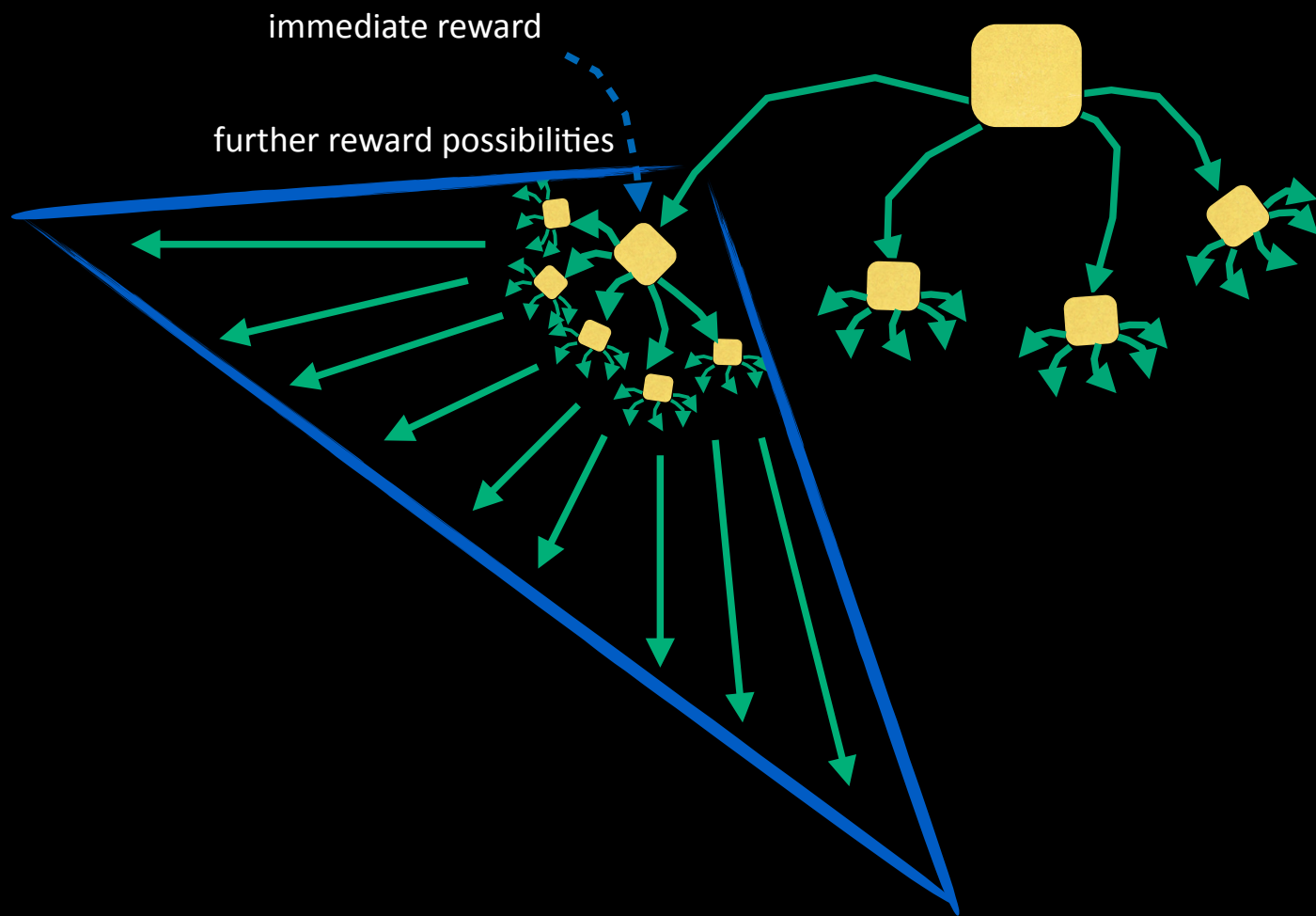
$\gamma=1$

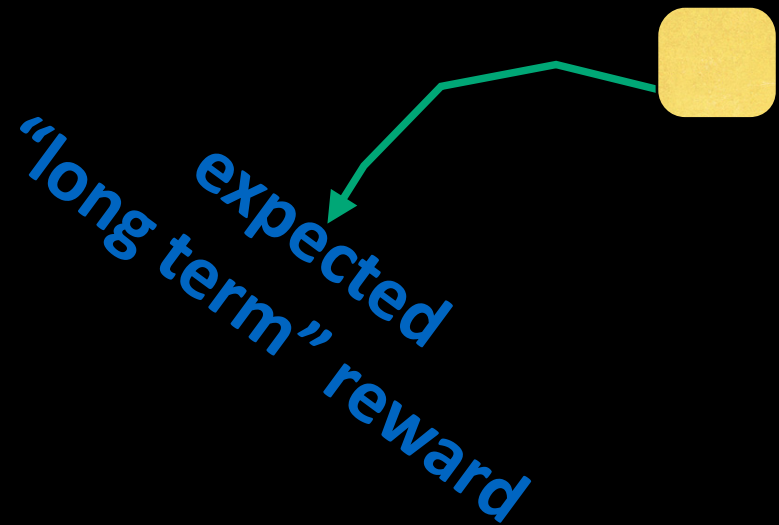
$$R_0 = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \gamma^4 r_5 + \dots$$

$$R_0 = \sum_{k=0}^T \gamma^k r_{k+1}$$

$$E \left\{ R_t = \sum_{k=0}^T \gamma^k r_{t+k+1} \right\}$$







$$E \left\{ R_t = \sum_{k=0}^T \gamma^k r_{t+k+1} \right\}$$

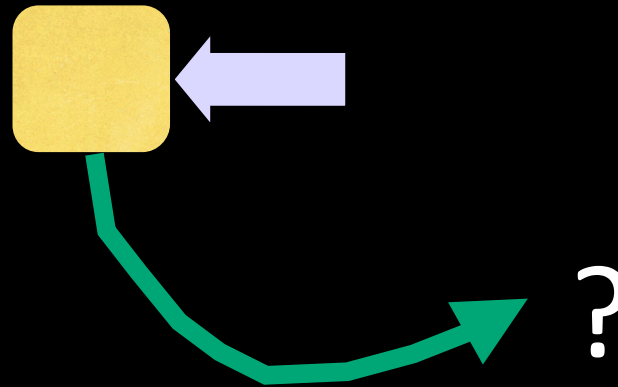
but these expected  
rewards are  
**not known to agent**  
beforehand!

whether they are known or not, the  
agent has to act somehow!

**how to act/action selection?**

**how to get to know/estimate?**

# action selection?



**values** of each possible action  
in the current state?

**expected reward** for  
carrying out the action is its **value**

**but what are  
these values?**

**<<expected rewards are not known>>  
<<actions based on expected rewards>>**

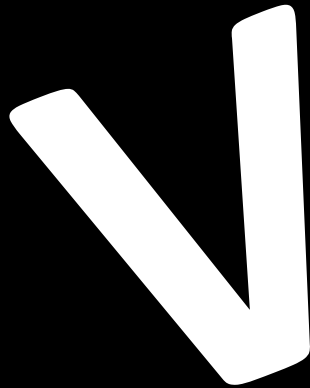
**these expected rewards  $E\{R_t\}$  are to be  
estimated by agent  
whilst acting!**

# Q

	a	b	c
1	2	0	1
2	3	0	-1
3	-5	6	2
4	2	3	1
.	.	.	.
.	.	.	.
.	.	.	.
n	7	8	7

agent maintains values  
for actions within each state

selects actions using these values based on a  
“policy”



1	2
2	3
3	-5
4	2
.	.
.	.
.	.
n	7

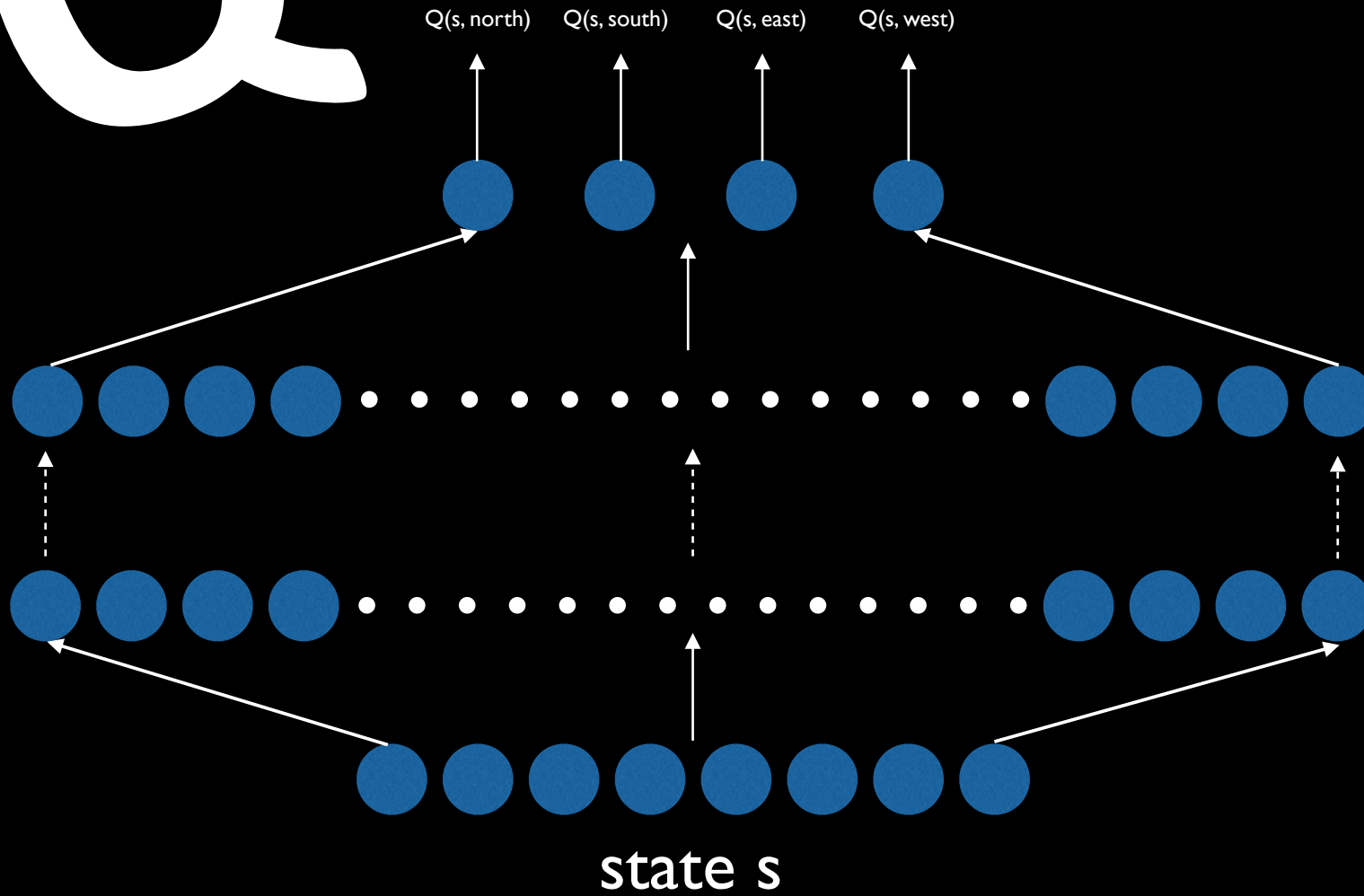
agent maintains state values

selects actions using these values based on a  
**“policy”**

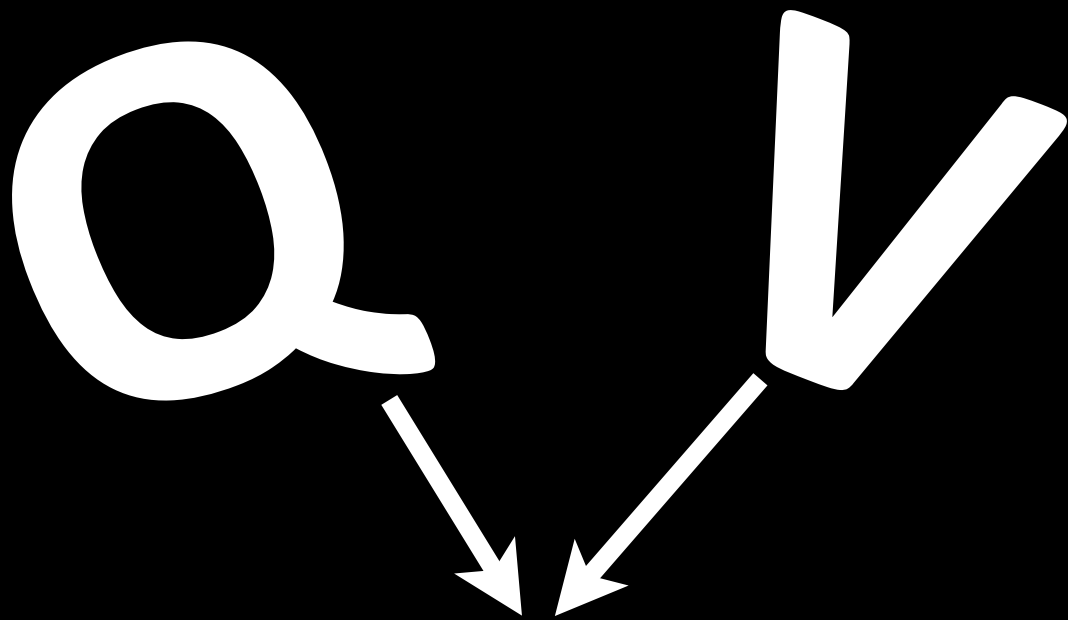


# Q

## action values given state



$Q$   $V$

A diagram showing two large white letters, 'Q' on the left and 'V' on the right, positioned at the top. Two white arrows originate from the bottom of 'Q' and the bottom of 'V', pointing downwards and towards each other to converge on the letter 'E' of the expression 'E{R\_t}' below.

$E\{R_t\}$

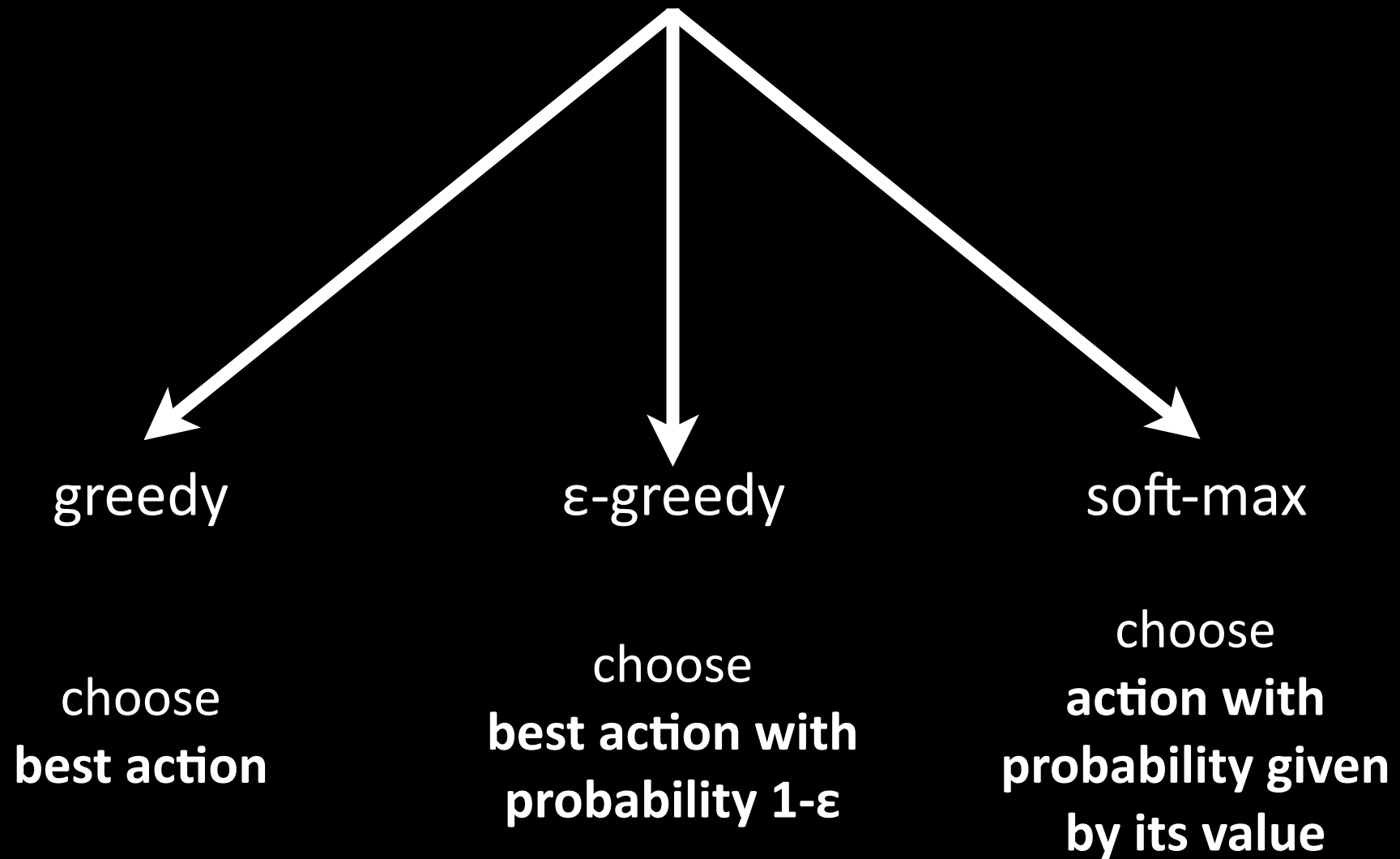
# policy?

probability of choosing  
an action, given a state

# $\pi$

 $Q^\pi(s, a)$  $V^\pi(s)$

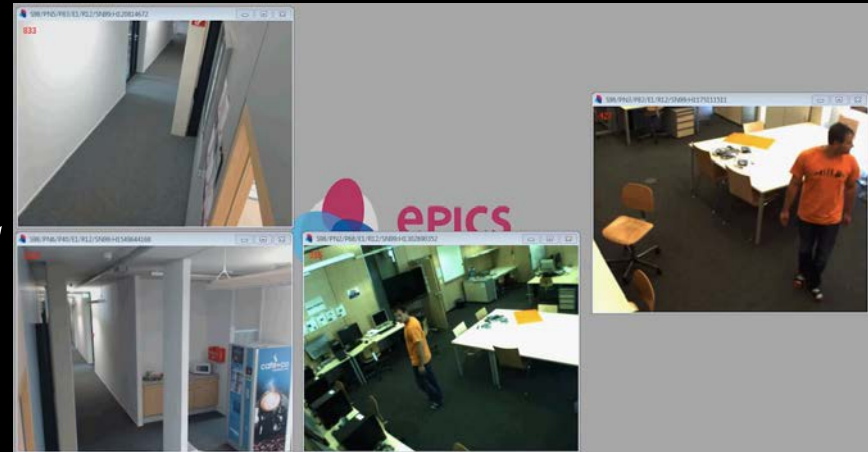
# usual policies



# exploration vs. exploitation policy = multi-armed bandit strategy



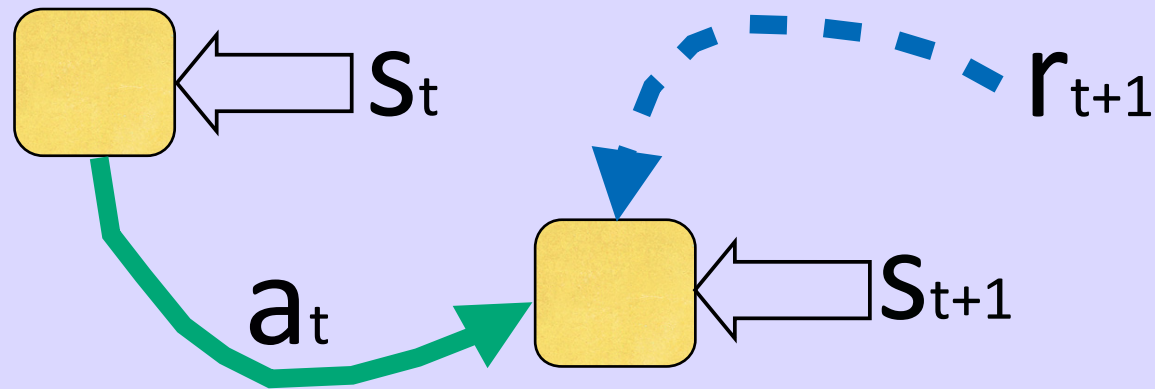
Yamaguchi先生, [http://en.wikipedia.org/wiki/File:Las\\_Vegas\\_slot\\_machines.jpg](http://en.wikipedia.org/wiki/File:Las_Vegas_slot_machines.jpg)



**Learning to be different: Heterogeneity and efficiency in distributed smart camera networks**, P. R. Lewis, L. Esterle, A. Chandra, B. Rinner, and X. Yao, In Proceedings of the IEEE Conference on Self-Adaptive and Self-Organizing Systems (SASO), IEEE, 2013.

**Static, dynamic and adaptive heterogeneity in socio-economic distributed smart camera networks**, P. R. Lewis, L. Esterle, A. Chandra, B. Rinner, J. Torresen, and X. Yao, ACM Transactions on Autonomous and Adaptive Systems (TAAS), ACM, 2015.

A/B Testing



## estimation?

<<use currently visible rewards to update values of where you are coming from>>

the current state (or state-action pair) has an **estimated value** (say zero/random initially), which can be used **together with  $r_{t+1}$**  to **update value** of previous state (or state-action pair)

i.e.

fraction of (currently visible rewards - old value)

+

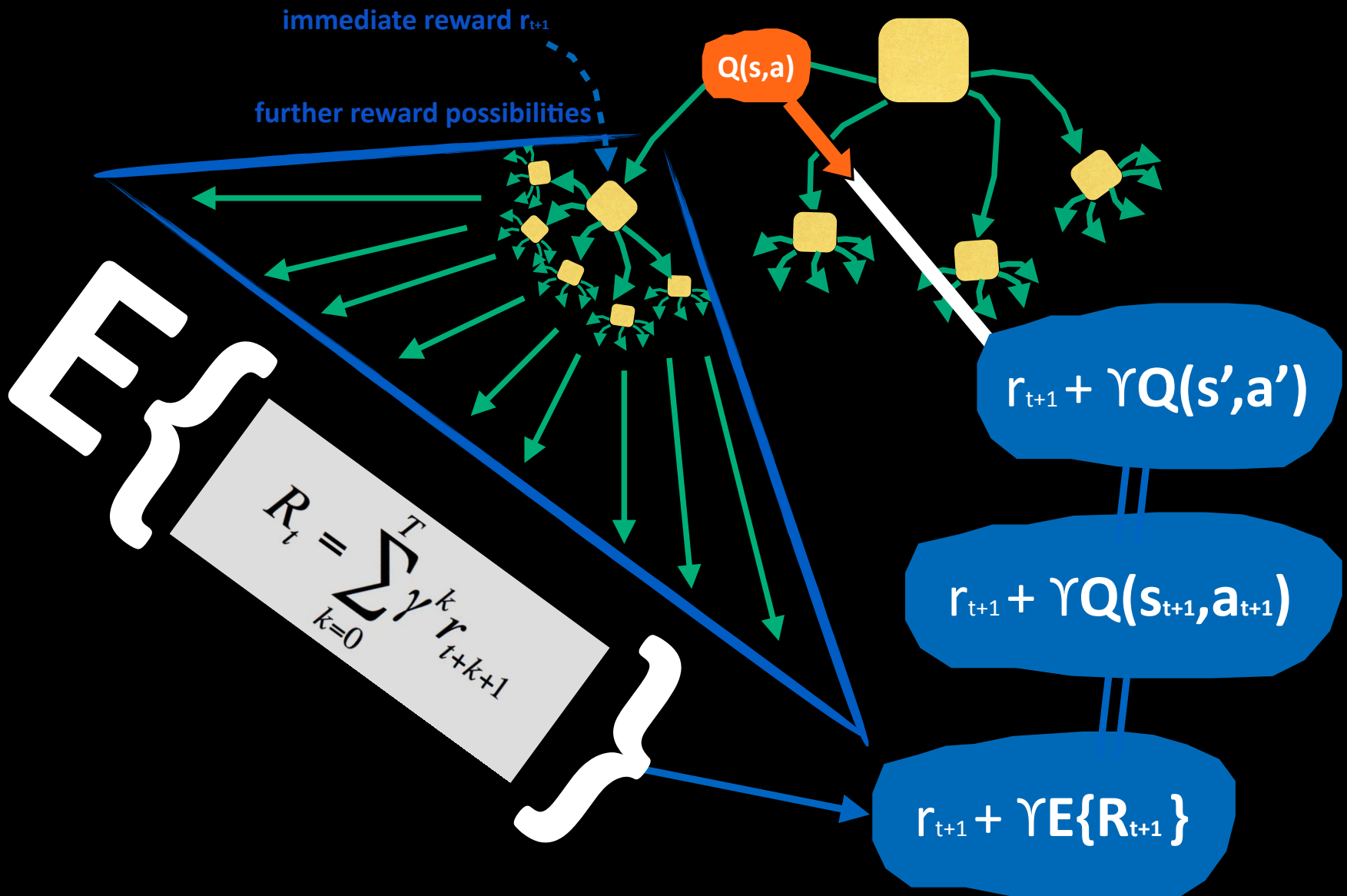
old value



new value

(1-fraction) old value + fraction curr. vis. rewards







$$V(s) \leftarrow V(s) + \mu(r + \gamma V(s') - V(s))$$

e.g.

$$Q(s, a) \leftarrow Q(s, a) + \mu(r + \gamma Q(s', a') - Q(s, a))$$

e.g. update  
a lookup table maintaing  
expected rewards

	a	b	c
1	2	0	1
2			-1
3	5		2
4	2	3	1
.	.	.	.
.	.	.	.
.	.	.	.
n	7	8	7

**Q**

1	2
2	3
.	5
4	2
.	.
.	.
.	.
n	7

**V**

$$Q(s, a) \leftarrow Q(s, a) + \mu(r + \gamma Q(s', a') - Q(s, a))$$

let's play with a version of the  
above update rule:

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

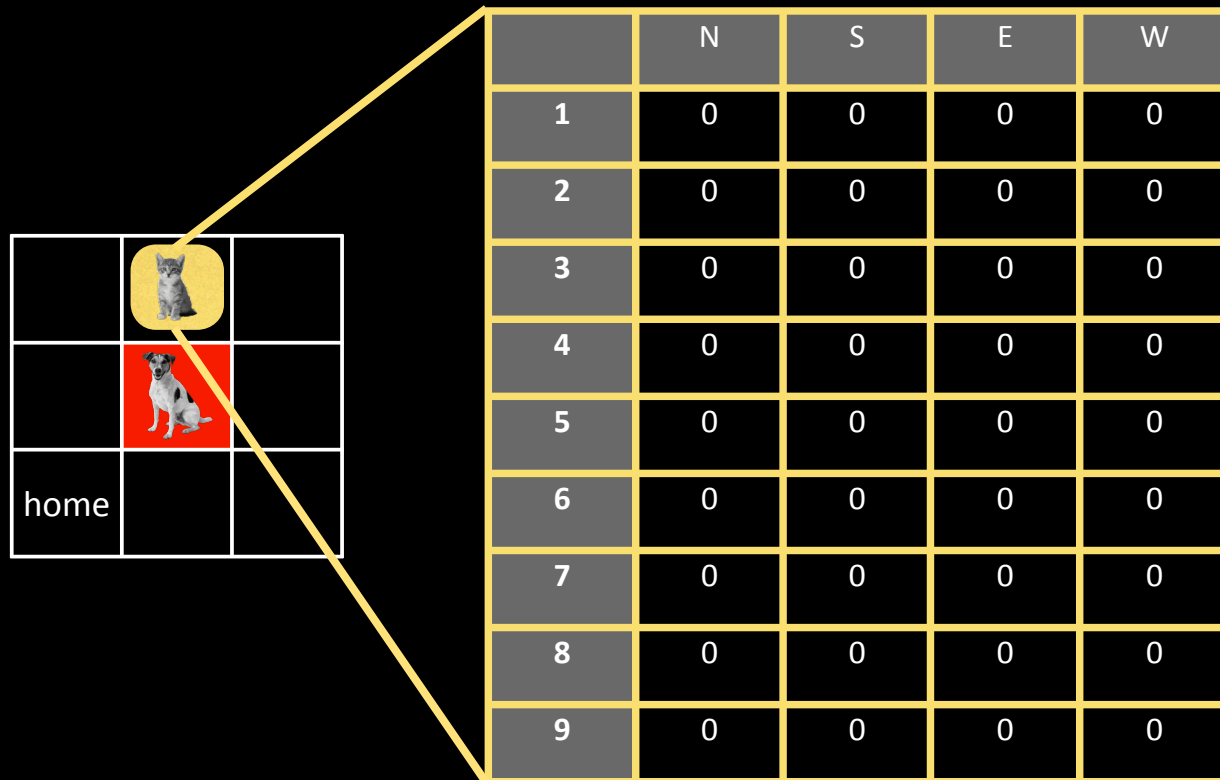
indicates  $a'$  to be the action  
with maximum value in next  
state  $s'$

let's play with a version of the  
above update rule:

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

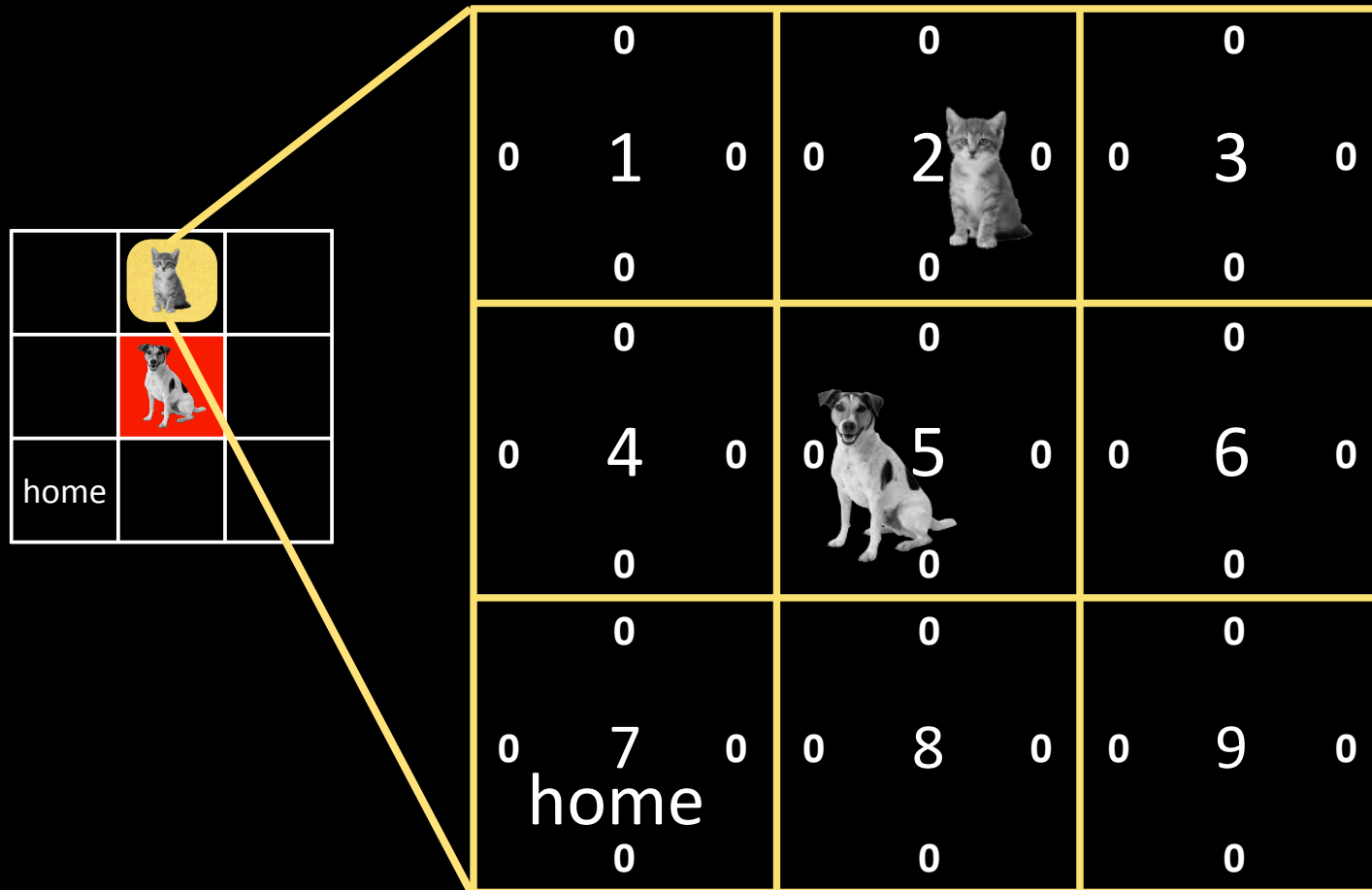
# our toy problem

## lookup table

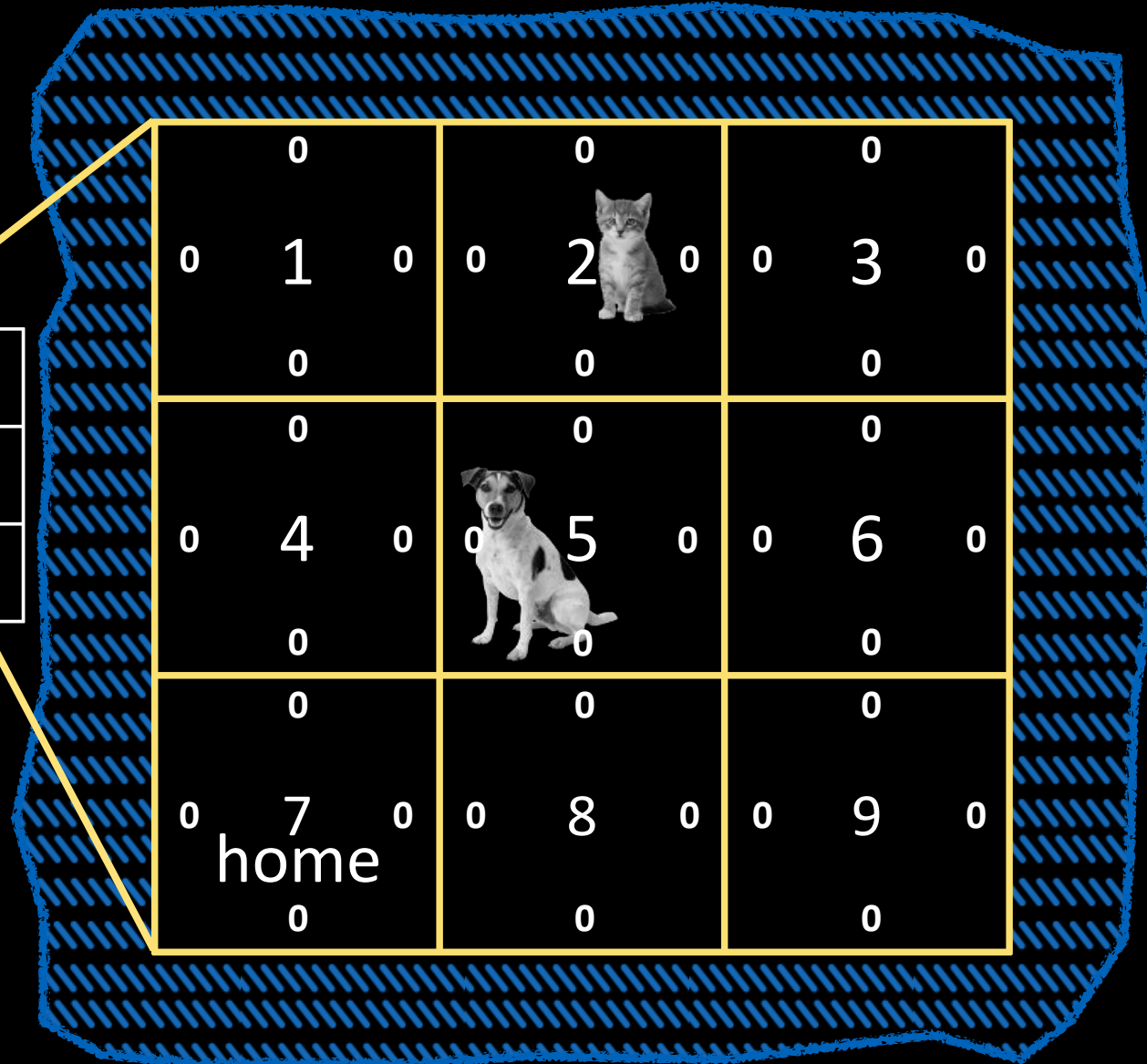
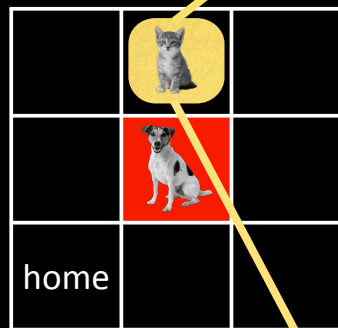


	N	S	E	W
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0

# our toy problem lookup table



reward structure?



**move...**

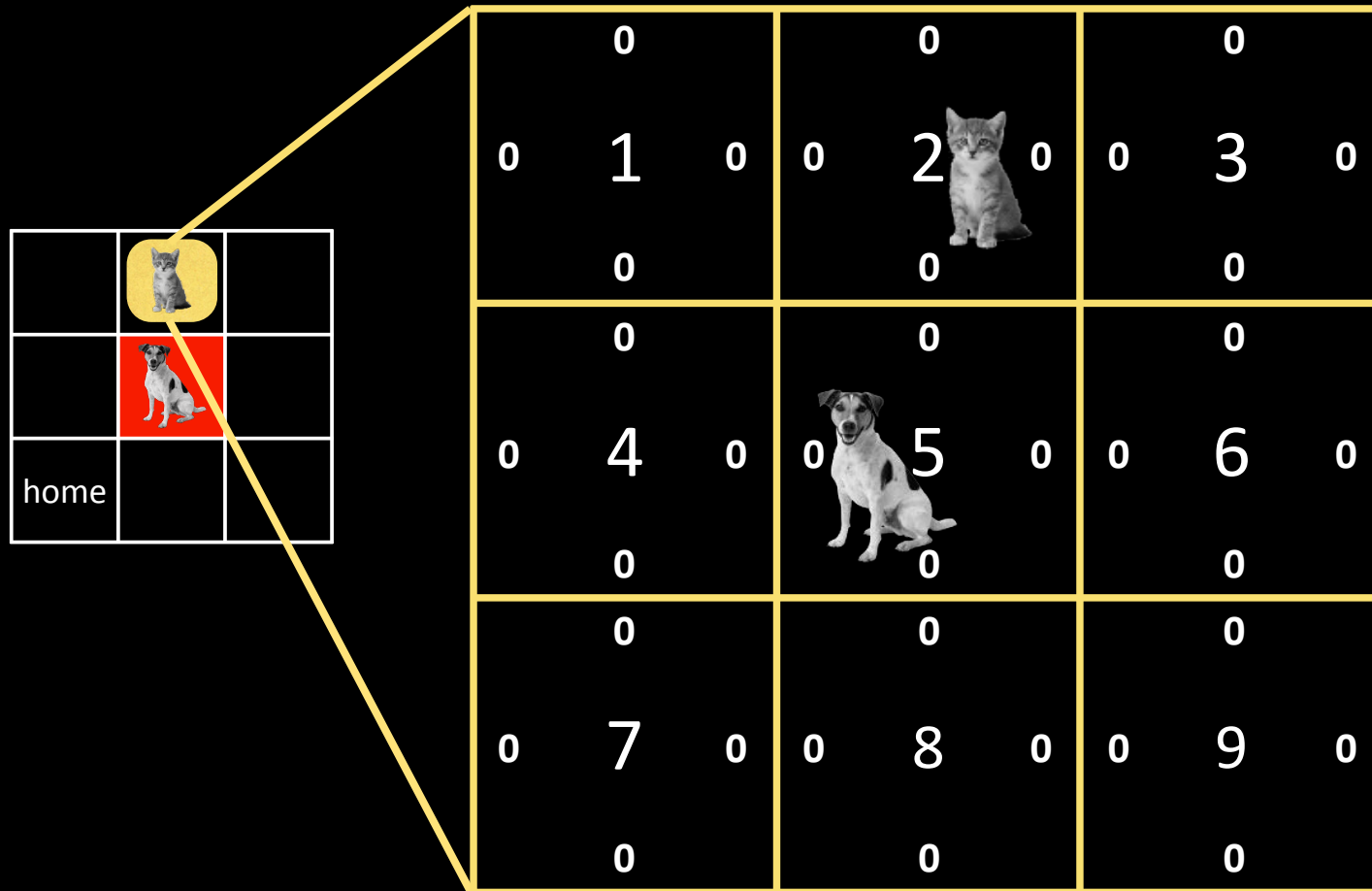
to any cell except 5 and 7:  
-1

out of bounds:  
-5

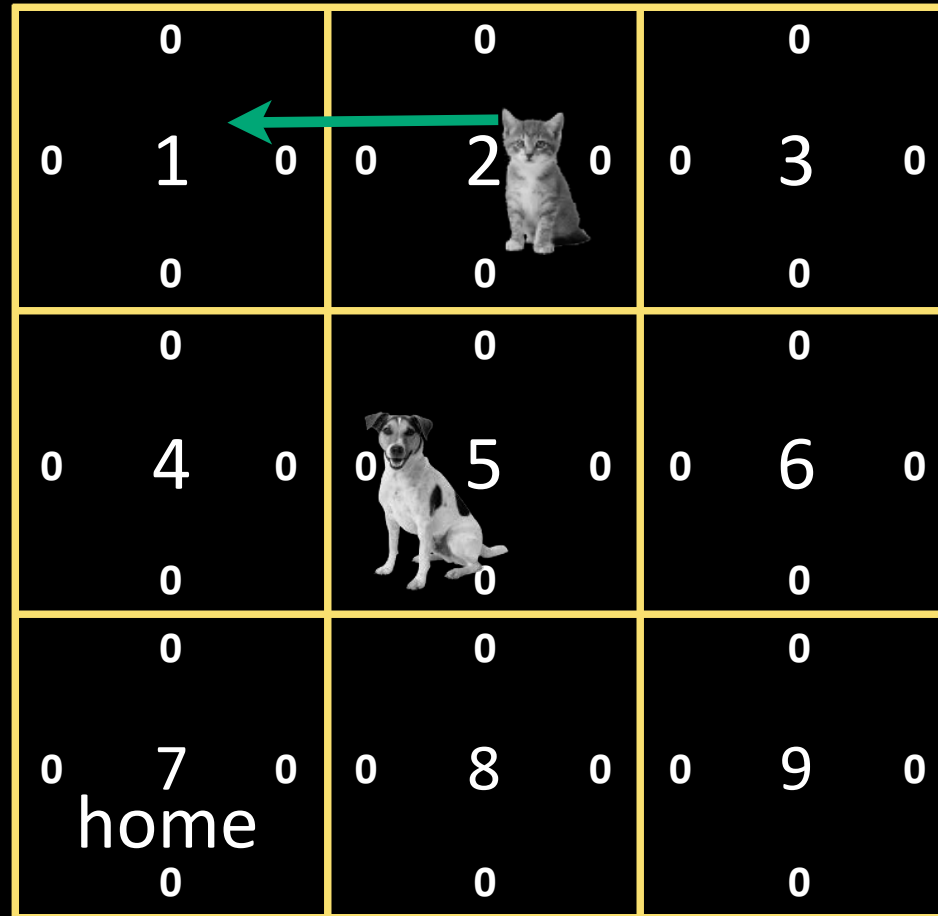
to 5:  
-10

to 7/home:  
10

let's fix  $\mu = 0.1$ ,  $\gamma = 0.5$








episode 1 begins...





0	0	0
0 1 0	-0.1 2 0	0 3 0
0	0	0
0	0	0
0 4 0	0 5 0	0 6 0
0	0	0
0 7 0	0 8 0	0 9 0
home	0	0
0	0	0





	?		0		0		0
0	1	0	-0.1	2	0	0	3
			0	0		0	0
0	0		0	0		0	0
0	4	0	0	5	0	0	6
			0	0		0	0
0	0		0	0		0	0
0	7	0	0	8	0	0	9
	home		0	0		0	0
	0		0	0		0	0



-5

-0.5	0	0
0 1 0 	-0.1 2 0	0 3 0
0	0	0
0 4 0	0 5 0 	0 6 0
0	0	0
0 7 0 home	0 8 0	0 9 0
0	0	0

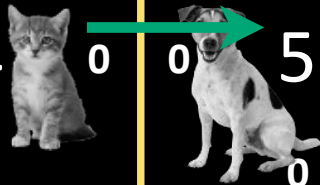
-0.5 0 1 0 0	0 -0.1 2 0 0	0 0 3 0 0
0 0 4 0 0	0 0 5 0 0	0 0 6 0 0
0 0 7 0 home 0	0 0 8 0 0	0 0 9 0 0

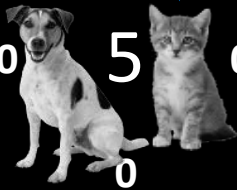
-0.5	0	0
0 1 0	-0.1 2 0	0 3 0
?	-1	0
0	0	0
0 4  0	0 5  0	0 6 0
0	0	0
0 7 0	0 8 0	0 9 0
home	0	0
0	0	0





-0.5	0	0
0 1 0	-0.1 2 0	0 3 0
-0.1	0	0
0	0	0
0 4  0	0 5  0	0 6 0
0	0	0
0 7 0	0 8 0	0 9 0
home		
0	0	0

	-0.5		0		0		0	
0	1	0	-0.1	2	0	0	3	0
	-0.1		0		0		0	
0	4	0	0	5	0	0	6	0
	0		0	0			0	
0	7	0	0	8	0	0	9	0
	home		0		0		0	
	0		0		0		0	





	-0.5		0		0		0	
0	1	0	-0.1	2	0	0	3	0
	-0.1			0		-10	0	
	0		0	0		0	0	
0	4	?	0	5	0	0	6	0
	0			0		0	0	
	0		0	0		0	0	
0	7	0	0	8	0	0	9	0
	home							
	0		0	0		0	0	

-0.5	0	0
0 1 0	-0.1 2 0	0 3 0
-0.1	0	0
0	0	0
0 4 -1	0  5 0	0 6 0
0	0	0
0 7 0	0 8 0	0 9 0
home	0	0
0	0	0

	-0.5		0		0		0	
0	1	0	-0.1	2	0	0	3	0
	-0.1		0		0		0	
	0		0		0		0	
0	4	-1	0	5	0	0	6	0
	0		0		0		0	
	0		0		0		0	
0	7	0	0	8	0	0	9	0
	home							
	0		0		0		0	

	-0.5		0		0		0	
0	1	0	-0.1	2	0	0	3	0
	-0.1		0		0		0	
	0		0		0		0	
0	4	-1	0	5	0	0	6	0
	0		?			-1	0	
	0		0		0		0	
0	7	0	0	8	0	0	9	0
home								
	0		0		0		0	

	-0.5		0		0		0	
0	1	0	-0.1	2	0	0	3	0
	-0.1		0		0		0	
	0		0		0		0	
0	4	-1	0	5	0	0	6	0
	0		 -0.1		0		0	
	0		0		0		0	
0	7	0	0	8	0	0	9	0
home								
0			0				0	

	-0.5		0		0		0	
0	1	0	-0.1	2	0	0	3	0
	-0.1		0		0		0	
	0		0		0		0	
0	4	-1	0	5	0	0	6	0
	0		 -0.1		0		0	
	0		0		0		0	
0	7	0	0	8	0	0	9	0
home								
	0		0		0		0	



	-0.5		0		0		0	
0	1	0	-0.1	2	0	0	3	0
	-0.1		0		0		0	
0	4	-1	0	5	0	0	6	0
	0			-0.1			0	
0	0		0	0			0	
0	7	0	?	8	0	0	9	0
	home							
	0						0	



home



10

?

0

-0.5	0	0
0 1 0	-0.1 2 0	0 3 0
-0.1	0	0
0	0	0
0 4 -1	0 5 0	0 6 0
0	 -0.1	0
0	0	0
0 7 0	1 8 0	0 9 0
home	0	0
0	0	0

episode 1 ends.

let's work out the next  
episode, starting at  
state 4

**go WEST and then SOUTH**

how does the table change?

	-0.5		0		0		0	
0	1	0	-0.1	2	0	0	3	0
	-0.1		0		0		0	
	0		0		0		0	
-0.5	4	-1	0	5	0	0	6	0
	1		-0.1		0		0	
	0		0		0		0	
0	7	0	1	8	0	0	9	0
	0		0		0		0	



and the next episode,  
starting at state 3

**go WEST -> SOUTH -> WEST -> SOUTH**

how does the table change?

	-0.5		0		0		0	
0	1	0	-0.1	2	0	-0.1	3	0
	-0.1		-1				0	
	0		0		0		0	
-0.5	4	-1	-0.05	5	0	0	6	0
	1.9		-0.1				0	
	0		0		0		0	
0	7	0	1	8	0	0	9	0
	0		0				0	



what we just saw was  
some episodes of  
**Q-learning**

value updates based on **optimal policy**:  
value of **best next action**

**off-policy learning**

# SARSA-learning?

value updates based on **used policy**:  
value of **the actual next action**

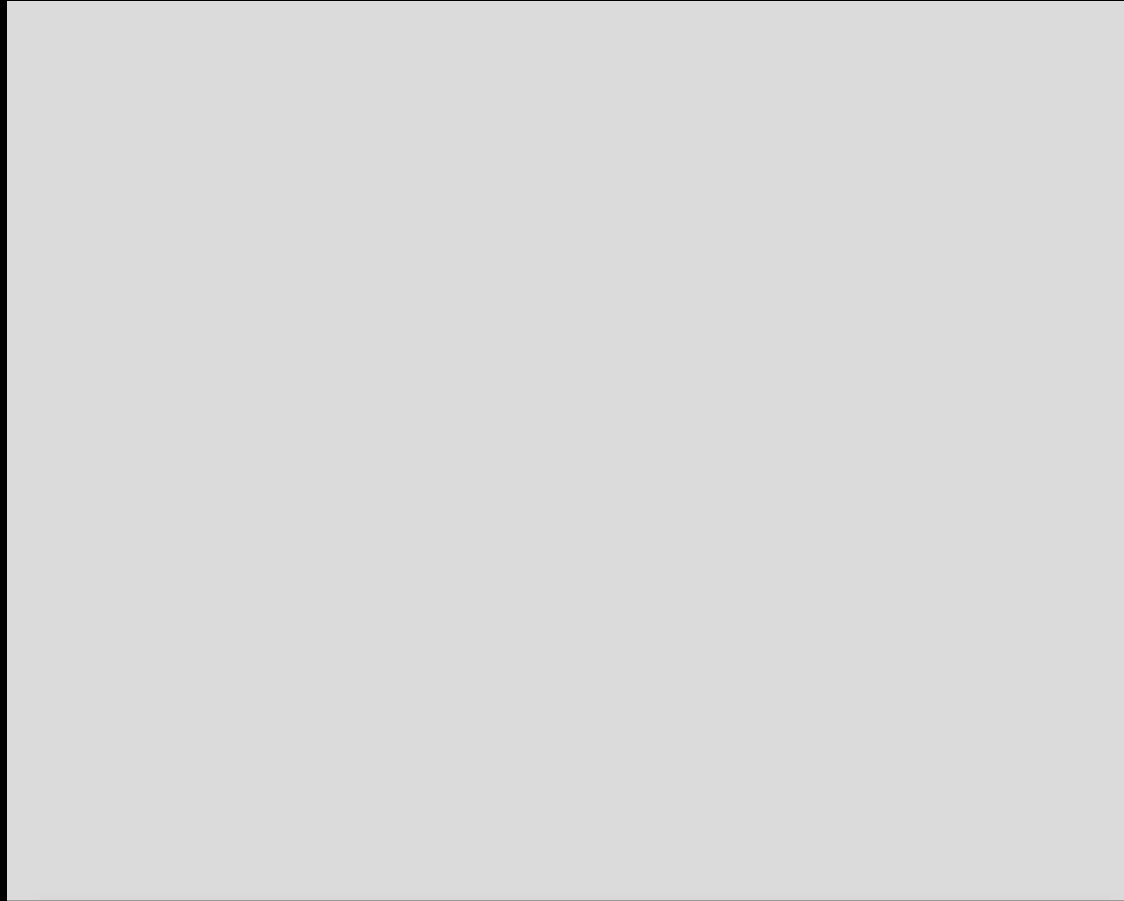
**on-policy learning**



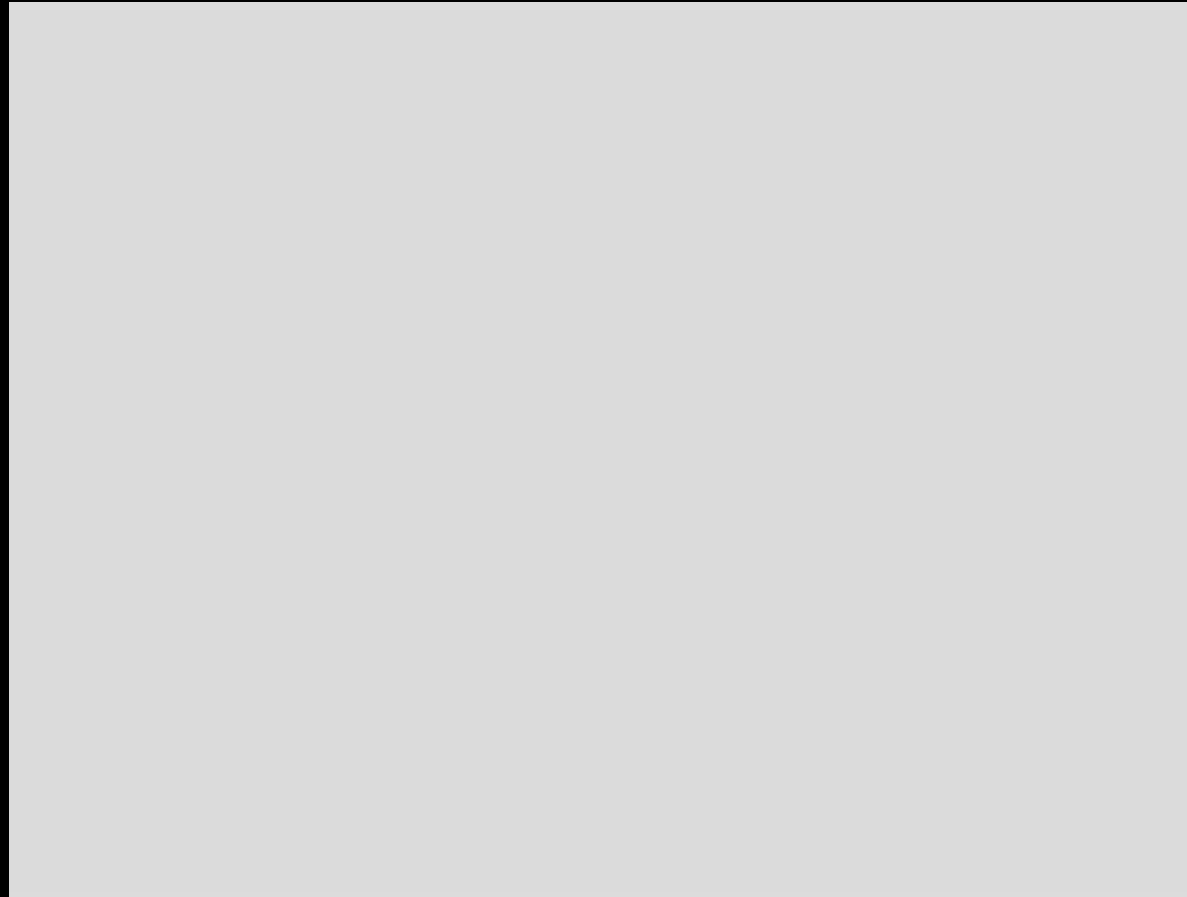


By Andreas Tille (Own work) [GFDL ([www.gnu.org/copyleft/fdl.html](http://www.gnu.org/copyleft/fdl.html)) or CC-BY-SA-3.0-2.5-2.0-1.0 ([www.creaCvecommons.org/licenses/by-sa/3.0/](http://www.creaCvecommons.org/licenses/by-sa/3.0/))], via Wikimedia Commons

mountain car...



# pole balancing...



Pole balancing in reality: <http://www.youtube.com/watch?v=Lt-KLtkDIh8>

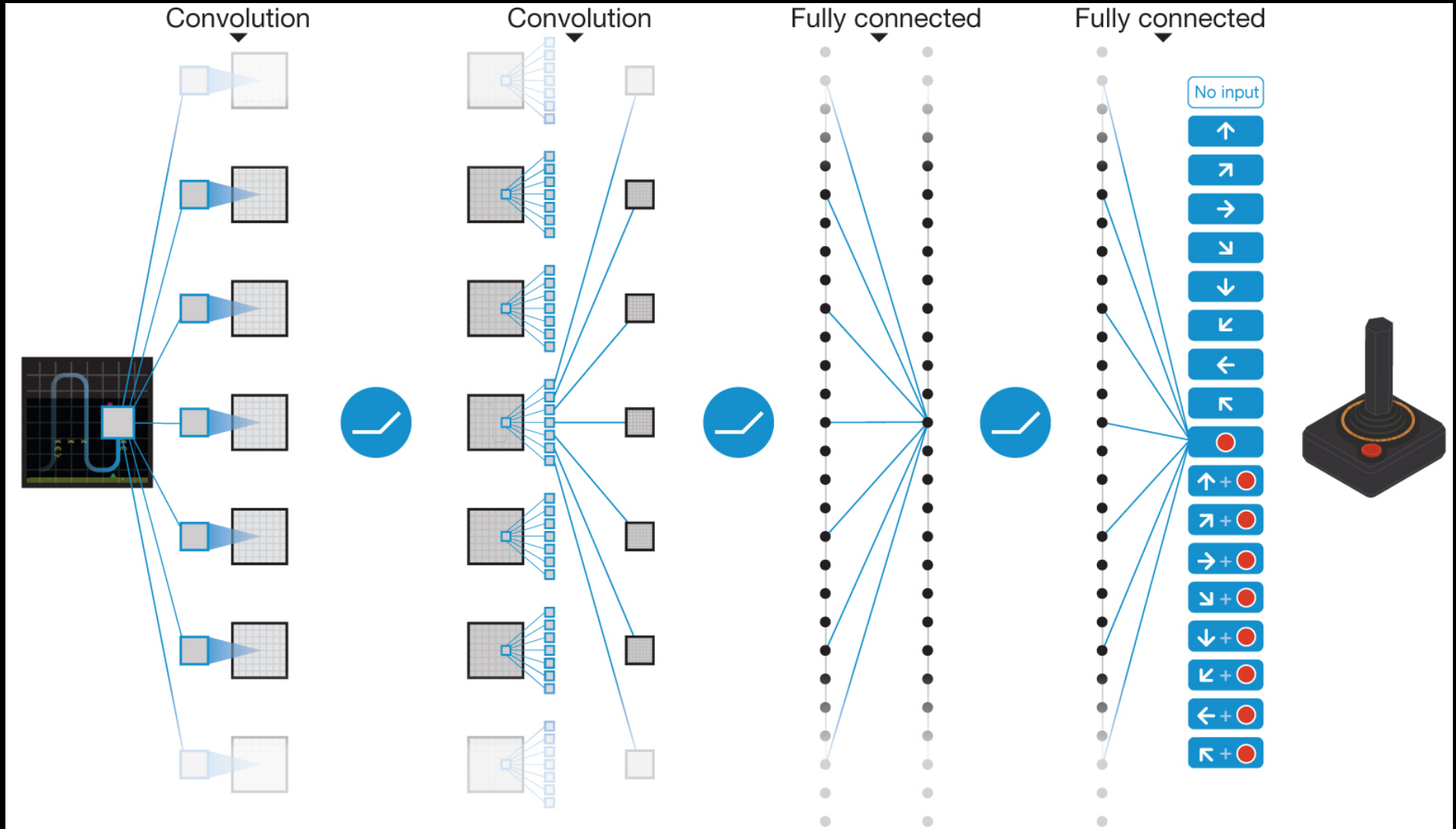
# human level game control

- **pixel** input
- **18 joystick/button positions** output
- **change in game score** as feedback
- **convolutional net representing Q**
- **backpropagation** for training!

Human-level control through deep reinforcement learning,  
Mnih et. al., Nature 518, Feb 2015

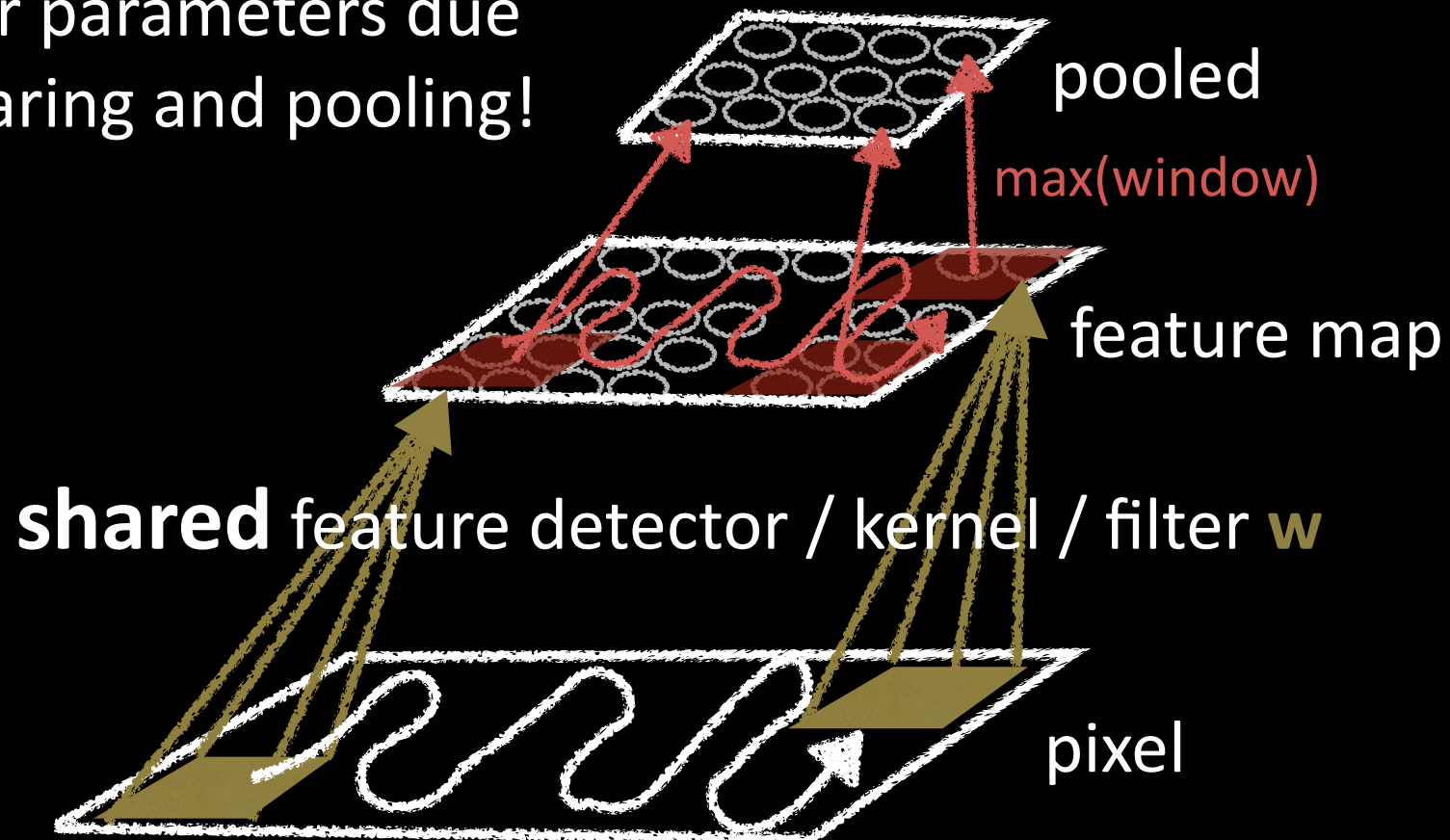
<http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html>

# neural network



# convolution, weight sharing, and pooling

fewer parameters due to sharing and pooling!



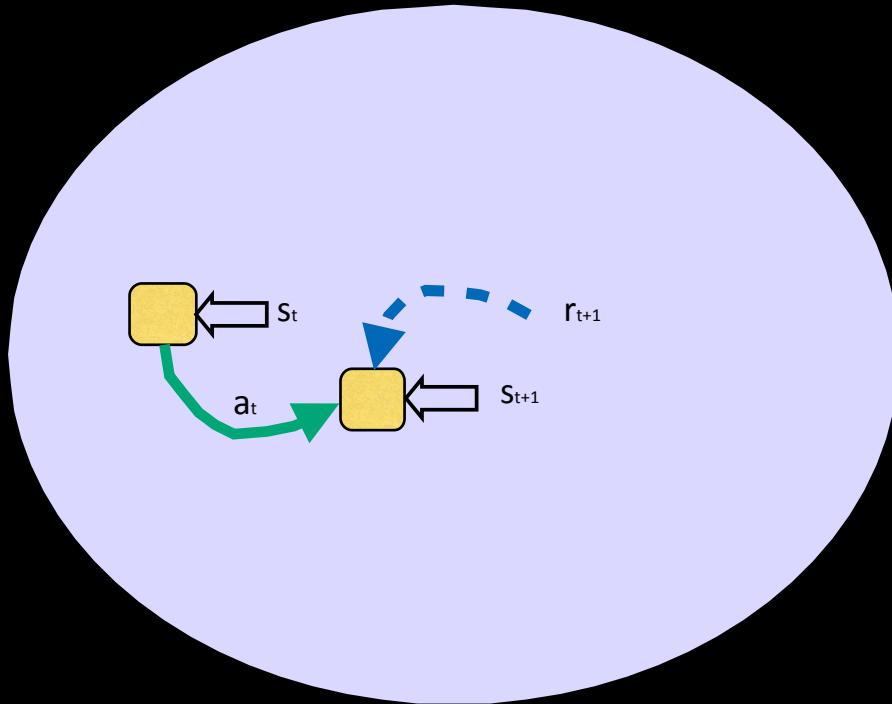
# backpropagation?

What is the **target** against which to minimise error?

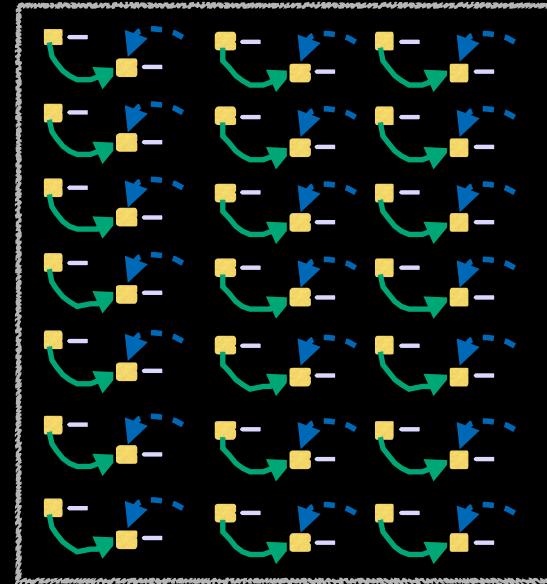
$$\mathcal{L}(w) = \mathbb{E} \left[ \left( \underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{\text{target}} - Q(s, a, w) \right)^2 \right]$$

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E} \left[ \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right) \frac{\partial Q(s, a, w)}{\partial w} \right]$$

# experience replay



**save current transition**  
( $s, a, r, s'$ ) in memory  
every time step



randomly sample a set of  
( $s, a, r, s'$ ) from memory  
for training Q network  
**(instead of learning from  
current state transition)**  
**every step**


= i.i.d + learn from the past



# freezing target Q

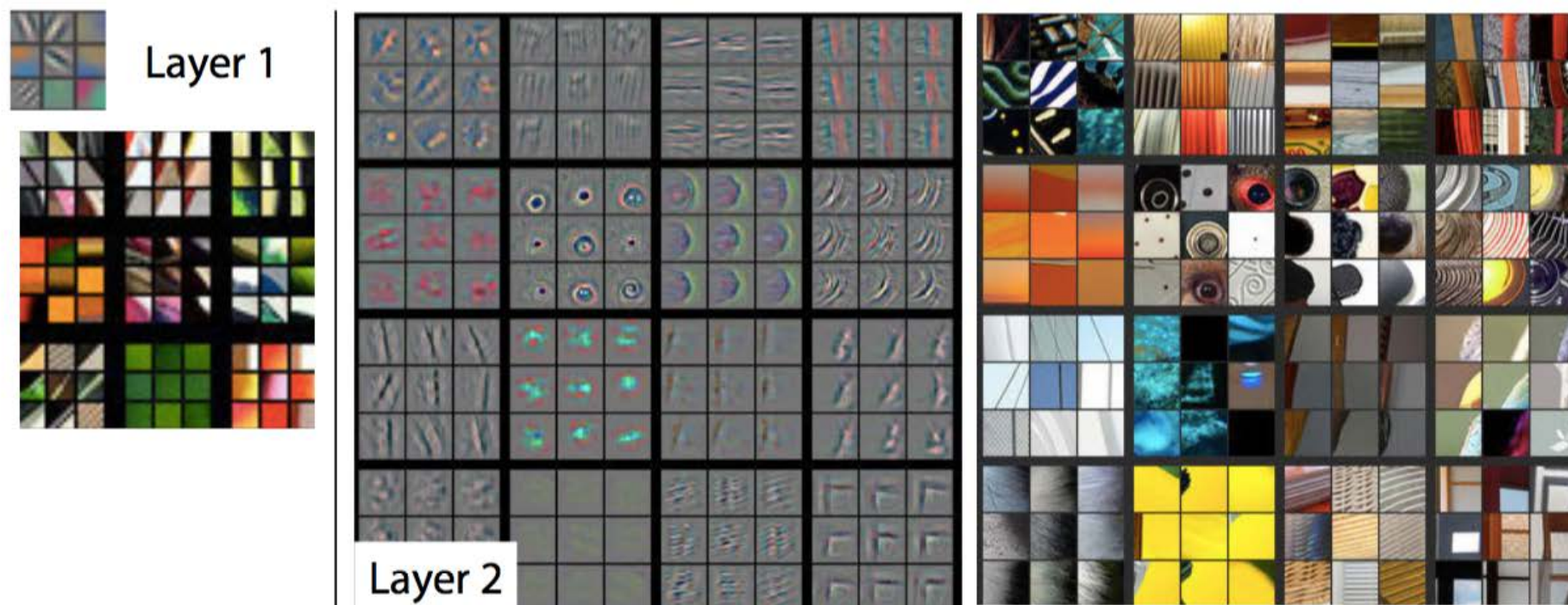
moving target => **oscillations**

freeze

$$\mathcal{L}(w) = \mathbb{E} \left[ \left( \underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{\text{target}} - Q(s, a, w) \right)^2 \right]$$


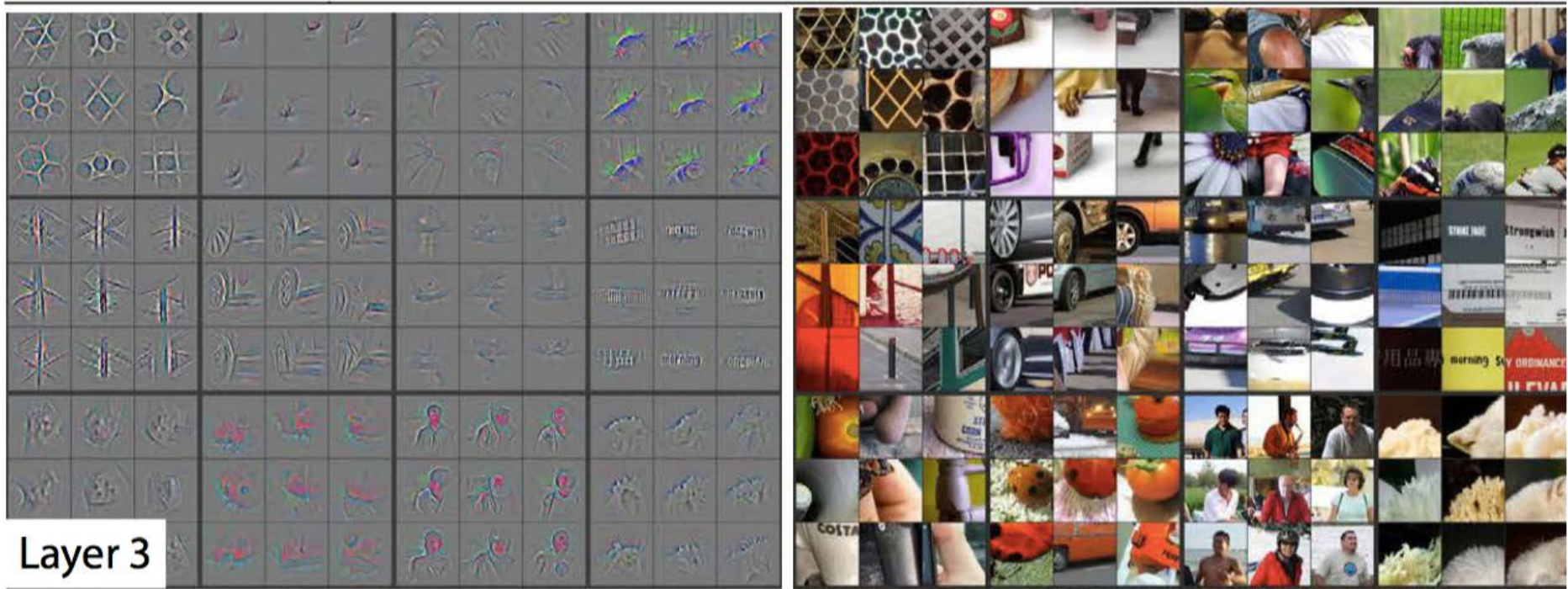
stabilise learning by **fixing target**,  
moving it every now and then

# what does a deep neural network do?

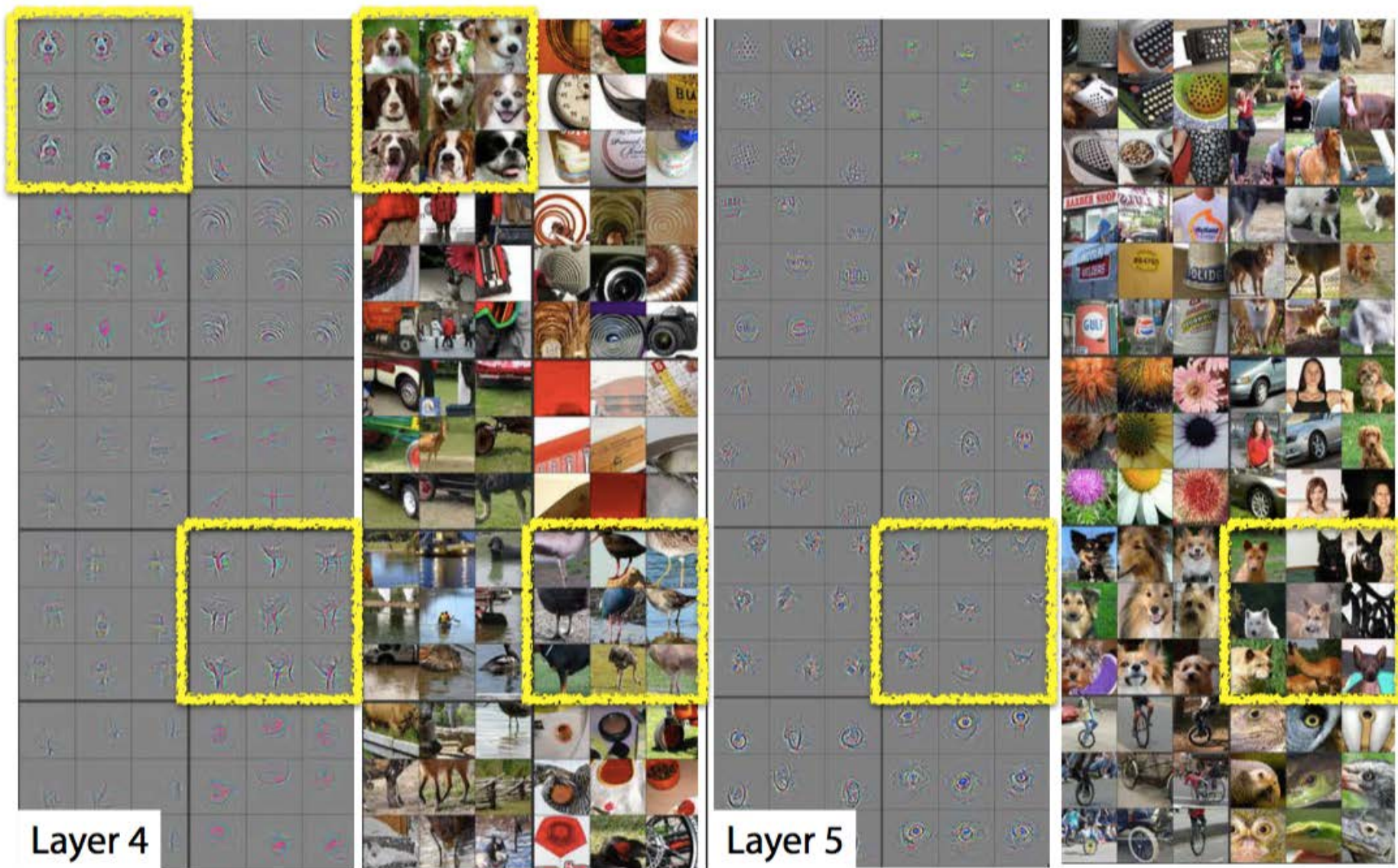


corners & edge/color conjunctions

reverse projections of neuron outputs in pixel space



similar textures



Layer 4

Object parts  
(dog face & bird legs)

Layer 5

Entire object with pose variation  
(dogs )

# compositional features

## compositional problem solving

multiplication (circuit design)

<— composed of adding numbers

<— composed of adding bits

```
output: x.y
——multiply——
——adding nums——
——adding bits——
input: x and y
```

## human knowledge organisation

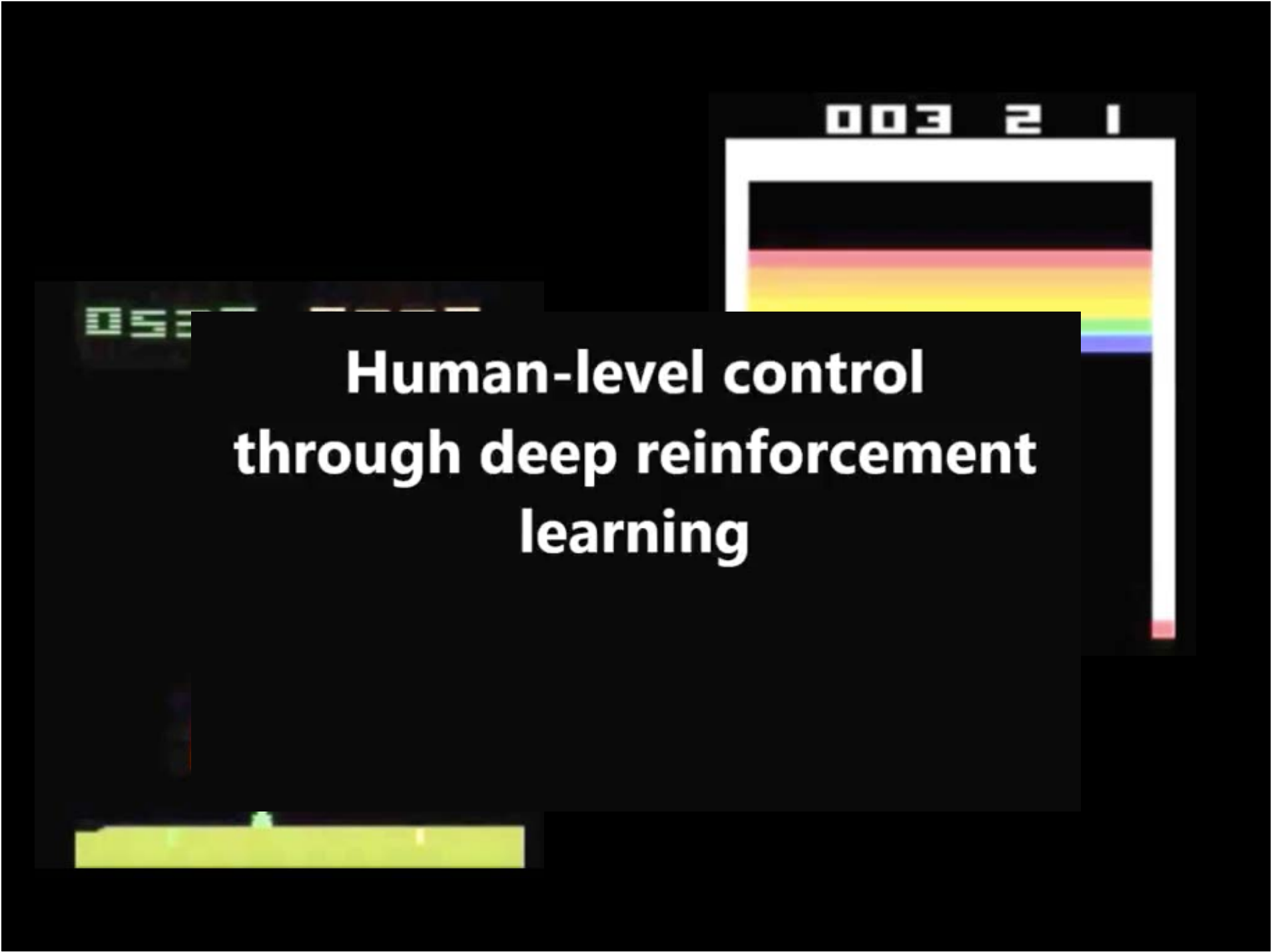
find roots of a linear expression

<— composed of setting expression to zero and solving linear equations

<— composed of rearranging terms

```
output: x = -2
——(find roots of x+2)——
——(set x+2=0)—(solve)——
——(rearrange)——
input: x, +, 2, =, 0
```

deep layers make representation of  
knowledge and processes happen with **fewer neurons!**



**Human-level control  
through deep reinforcement  
learning**

# code for you to play with...

tabular approaches:

[http://jamh-web.appspot.com/  
download.htm#Reinforcement Learning:](http://jamh-web.appspot.com/download.htm#Reinforcement_Learning)

deep learning approach:

Environment: <http://www.arcadelearningenvironment.org/>

Code: <https://sites.google.com/a/deepmind.com/dqn/>

please do e-mail for questions, and if you want to work on  
reinforcement learning research projects:

**arjun.chandra@gmail.com / chandra@ifi.uio.no**

# coyote learning what not to do...

