Reinforcement Learning

Coverage of Q-learning follows this blog post…

https://adventuresinmachinelearning.com/reinforcement-learning-tutorial-python-keras/

The discussion of A3C and Cart-pole follows….

https://medium.com/tensorflow/deep-reinforcement-learning-playing-cartpole-through-asynchronous-advantage-actor-critic-a3c-7eab2eea5296

Learning from interactions

Reinforcement learning (RL) is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal.

- *Reinforcement Learning*, Sutton and Barto (2012)

Key concepts:

Trial-and-error search

Delayed rewards

Exploration vs. exploitation

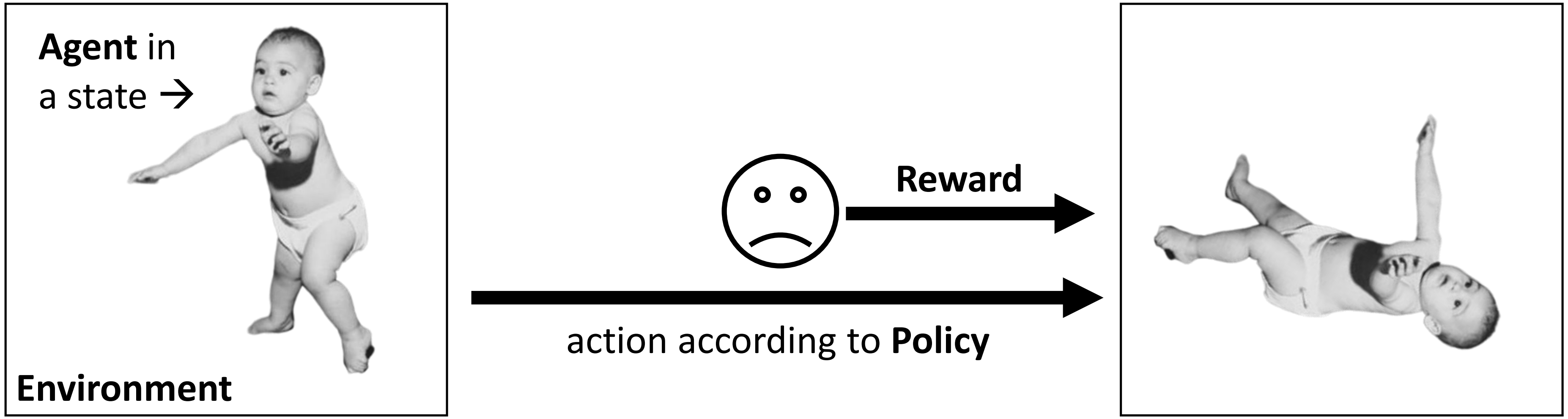
Posterazzi @ Walmart

Vocabulary in RL

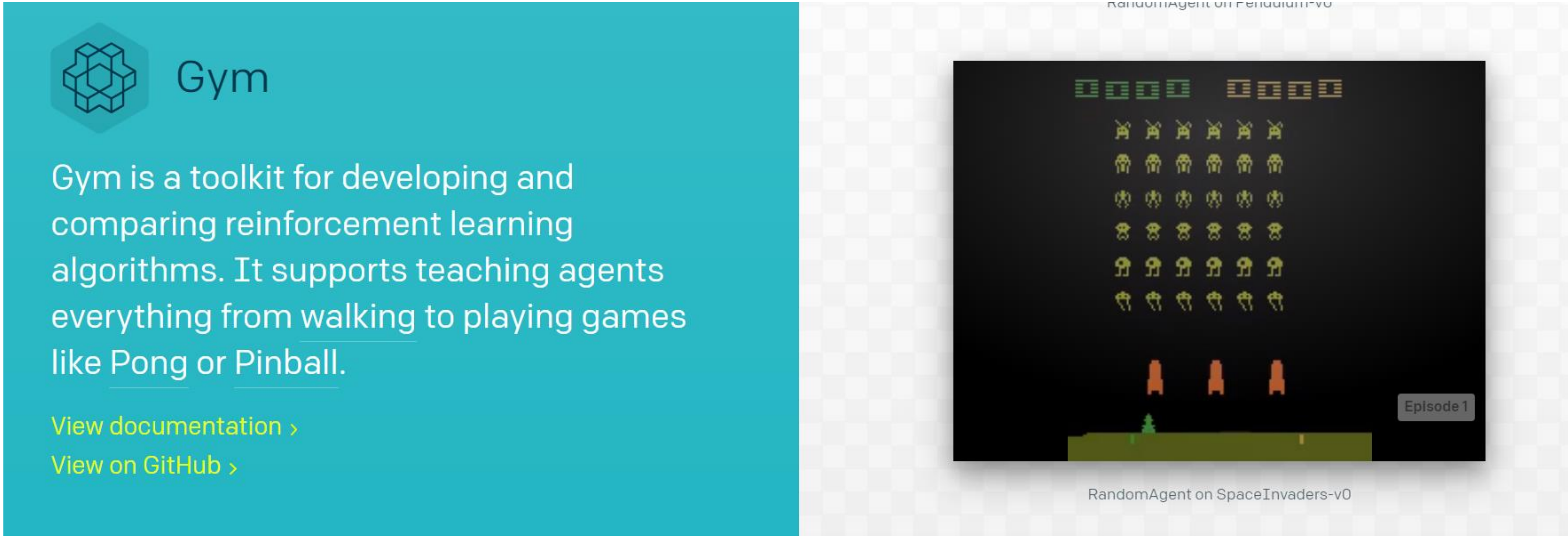
The **agent** (our baby) exists in a state within its **environment** (the room).

It takes actions (like trying to balance itself) according to a **policy**.

The action moves the agent to the next state and a **reward** is given (such as pleasure from staying upright or pain from falling).



Find environments at https://gym.openai.com/

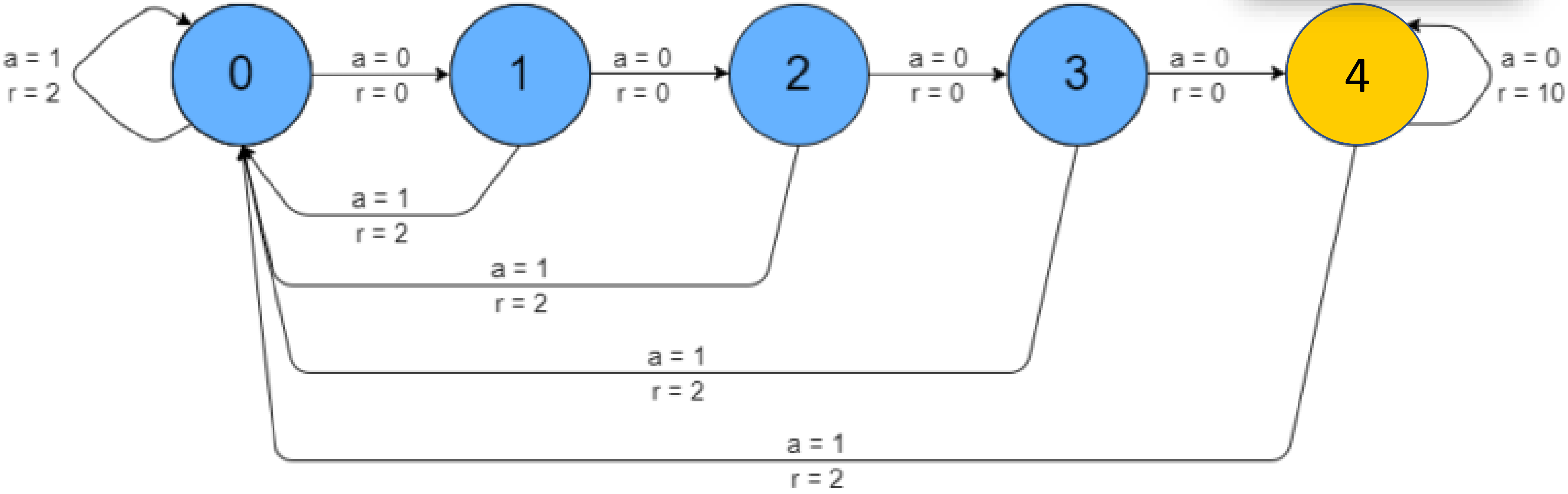


Example 1: Nchain

Two actions (step forward / all-the-way backward).

Delayed rewards!

Some random slips.



Q-learning: Attempt 1 (Naïve heuristic)

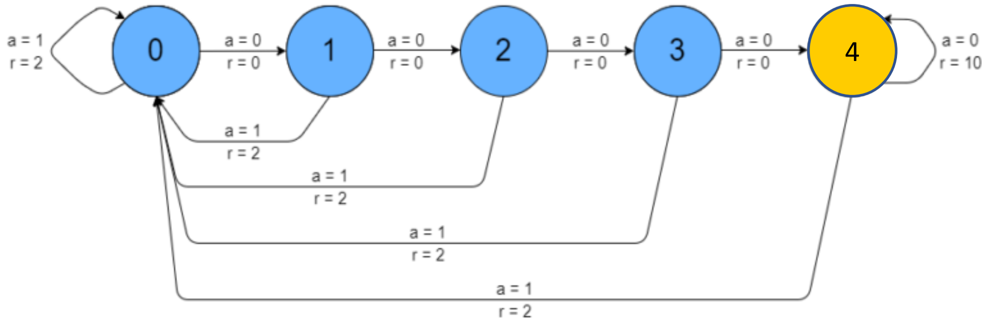
A (bad) policy

- $Q(s, a) = Q(s, a) + r$
- Agent chooses its action based on the sum of all previous rewards

	Move forward (0)	Move backward (1)
State 0	0	639006
State 1	0	129034
State 2	0	25418
State 3	0	4944
State 4	**0**	2762

Problems

- No delayed rewards
- No exploration



Q-learning: Attempt 2 (Delayed rewards)

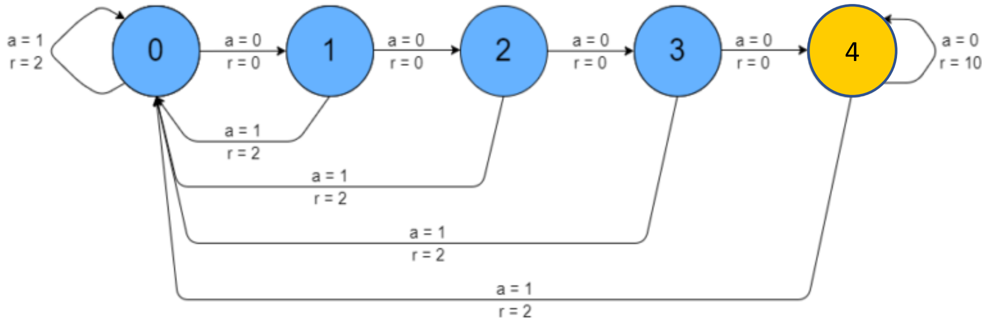
A (better) policy

- Q-learning:

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

- Delayed rewards! γ is called the discounting factor and the amount we replace Q with future rewards
- α is the learning rate (what percent of Q do we update with each iteration?)

	Move forward (0)	Move backward (1)
State 0	0	29.66
State 1	0	29.84
State 2	0	27.67
State 3	28.95	0
State 4	**0**	31.20

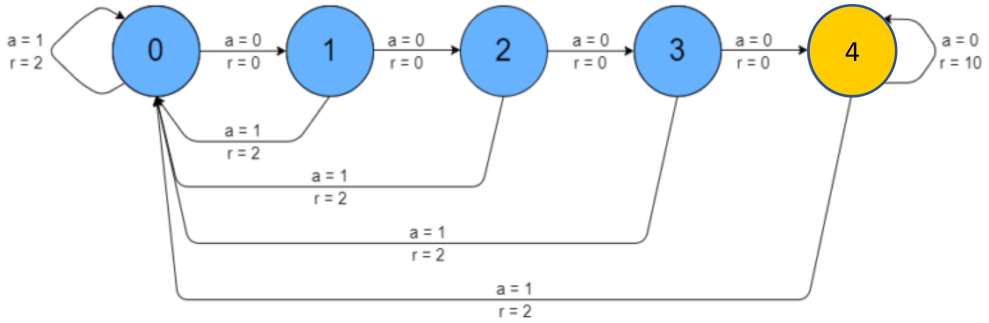


Q-learning: Attempt 3 (ε-greedy)

A (better) policy

- Exploration! Randomly choose the action sometimes
- Decay this randomness over time (simulated annealing)

	Move forward (0)	Move backward (1)
State 0	41.21	35.97
State 1	43.56	37.47
State 2	41.56	42.67
State 3	41.61	37.84
State 4	**59.73**	42.22



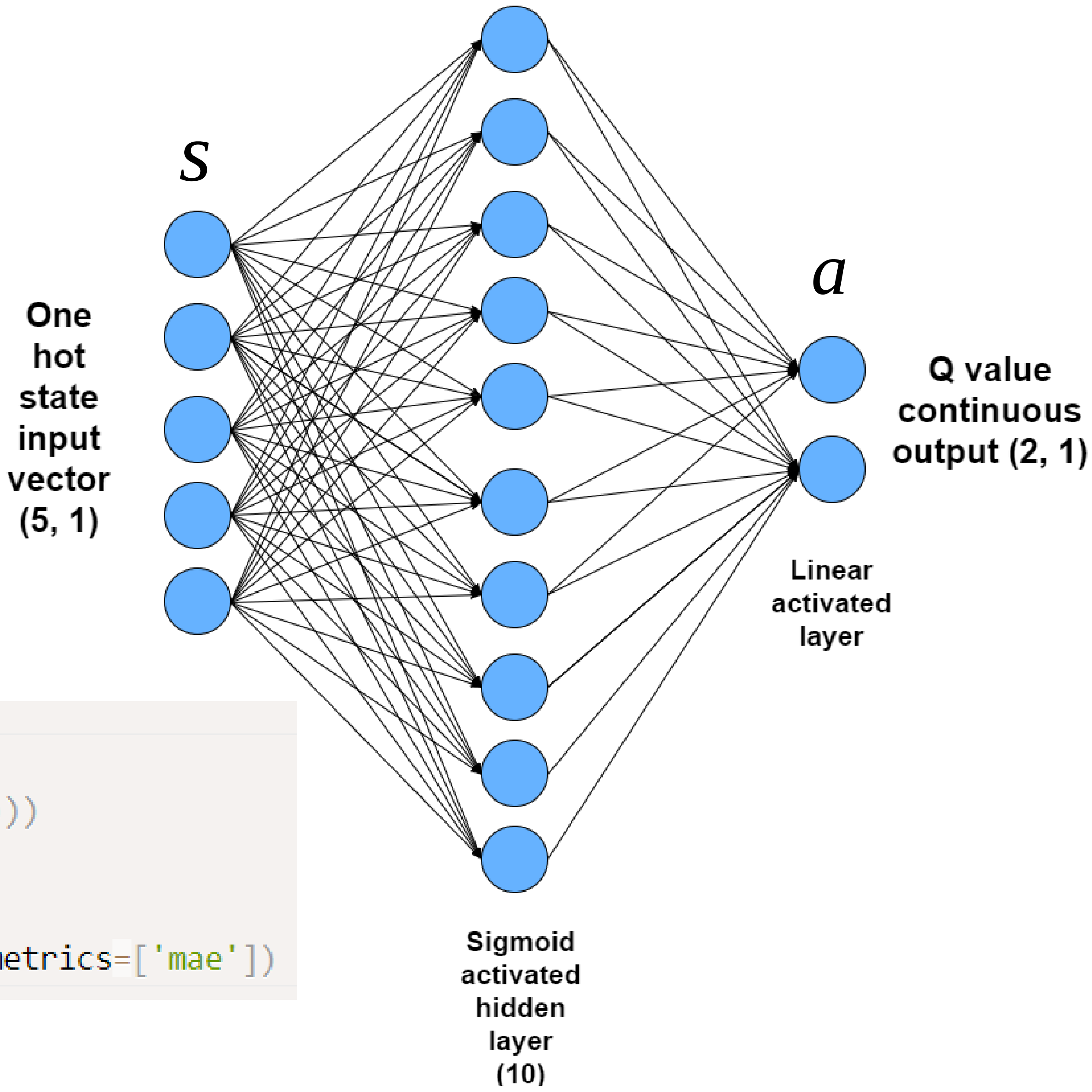
Problem

- That table might get huge.

Finally, a neural network

$$\mathscr{L}(\theta) = (\underbrace{r+\gamma\max_{a'} Q(s',a'|\theta)}_{\text{target}} - \underbrace{Q(s,a|\theta)}_{\text{prediction}})^2$$

```python
model = Sequential()
model.add(InputLayer(batch_input_shape=(1, 5)))
model.add(Dense(10, activation='sigmoid'))
model.add(Dense(2, activation='linear'))
model.compile(loss='mse', optimizer='adam', metrics=['mae'])
```



```python
# now execute the q learning
y = 0.95
eps = 0.5
decay_factor = 0.999
r_avg_list = []
for i in range(num_episodes):
    s = env.reset()
    eps *= decay_factor
    if i % 100 == 0:
        print("Episode {} of {}".format(i + 1, num_episodes))
    done = False
    r_sum = 0
    while not done:
        if np.random.random() < eps:
            a = np.random.randint(0, 2)
        else:
            a = np.argmax(model.predict(np.identity(5)[s:s + 1]))
        new_s, r, done, _ = env.step(a)
        target = r + y * np.max(model.predict(np.identity(5)[new_s:new_s + 1]))
        target_vec = model.predict(np.identity(5)[s:s + 1])[0]
        target_vec[a] = target
        model.fit(np.identity(5)[s:s + 1],
                  target_vec.reshape(-1, 2),
                  epochs=1,
                  verbose=0)
        s = new_s
        r_sum += r
    r_avg_list.append(r_sum / 1000)
```

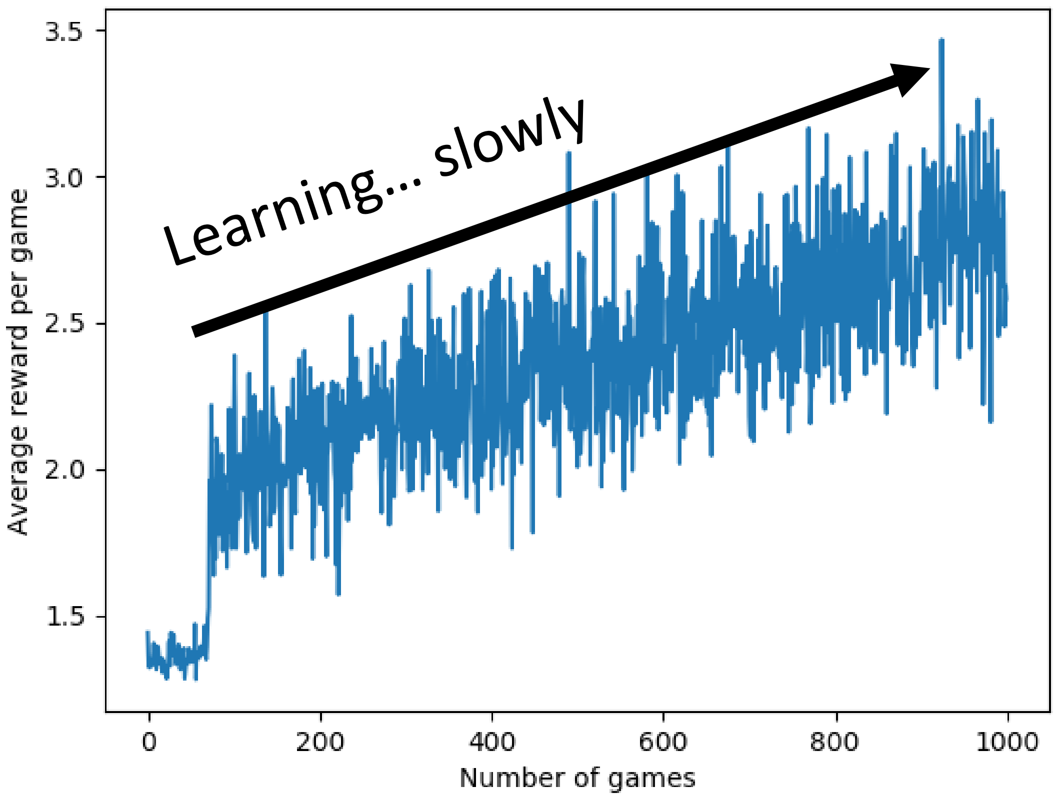
discounting factor

ε-greedy

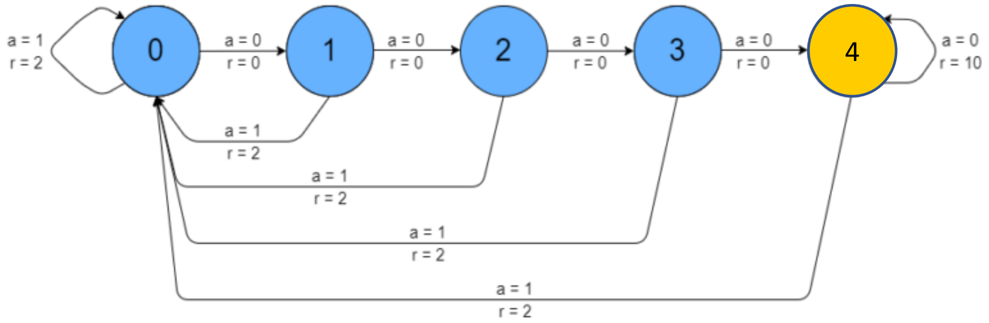
Fit the network

$$\mathcal{L}(\theta) = (\underbrace{r + \gamma \max_{a'} Q(s',a'|\theta)}_{\text{target}} - \underbrace{Q(s,a|\theta)}_{\text{target vec}})^2$$

Example 1: Results



	Move forward (0)	Move backward (1)
State 0	61.18	60.26
State 1	64.52	61.16
State 2	69.10	62.10
State 3	75.03	63.53
State 4	**83.42**	65.11



That was slow.
An introduction to A3C

Asynchronous

Multiple agents (threads) are used.

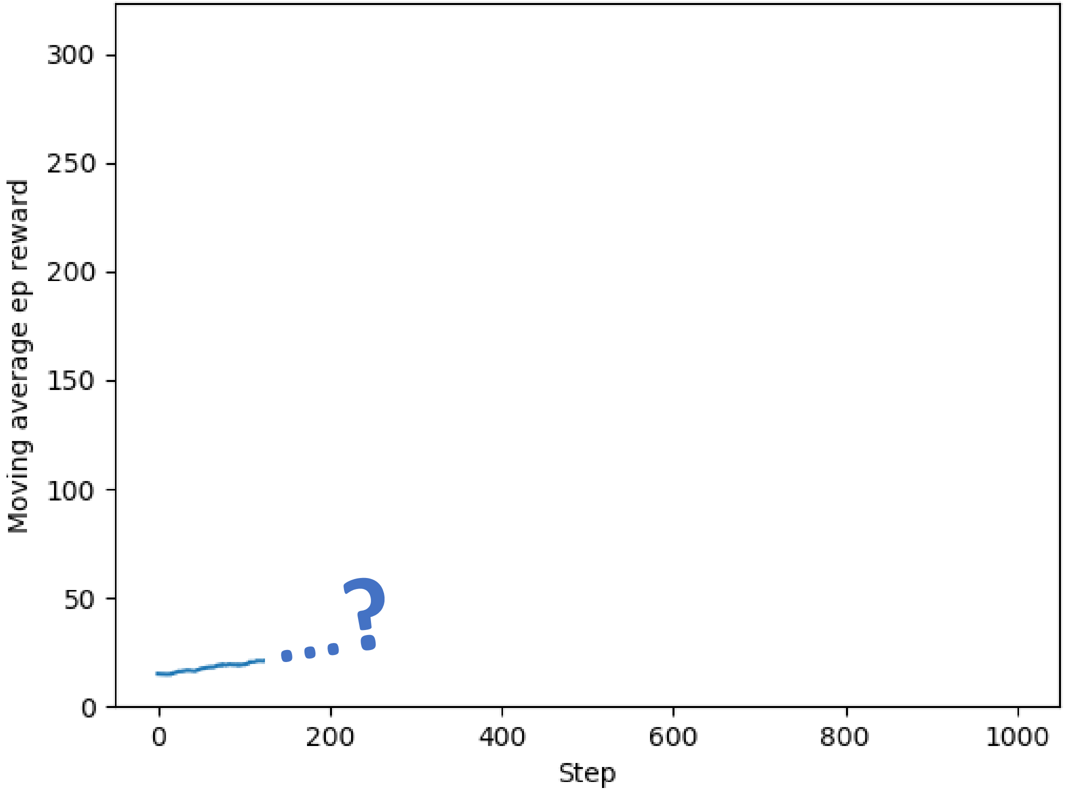
Advantage

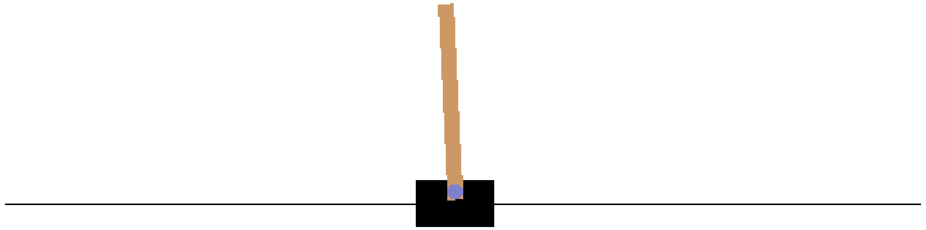
We value an action according to the advantage of following the policy π for all future actions.

Actor-Critic

Interchangeable with Q-learning (just another algorithm).

Example 2: Cart-pole (movie 1)





Example 2: Cart-pole (movie 2)