Yandex
Data Factory

LAMBDA

AgentNet

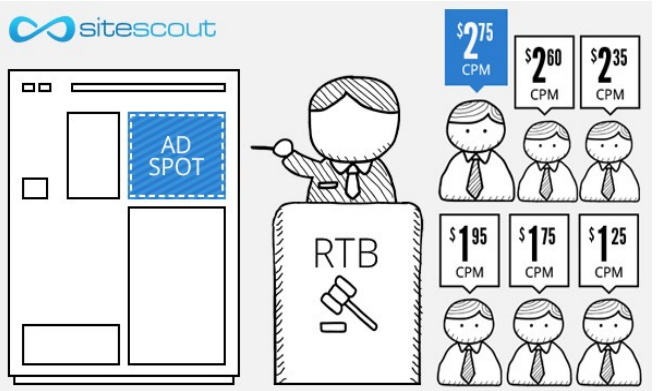
Deep reinforcement learning for humans

Alexander Panin, Alexey Rogozhnikov, Andrey Ustyuzhanin

Why bother

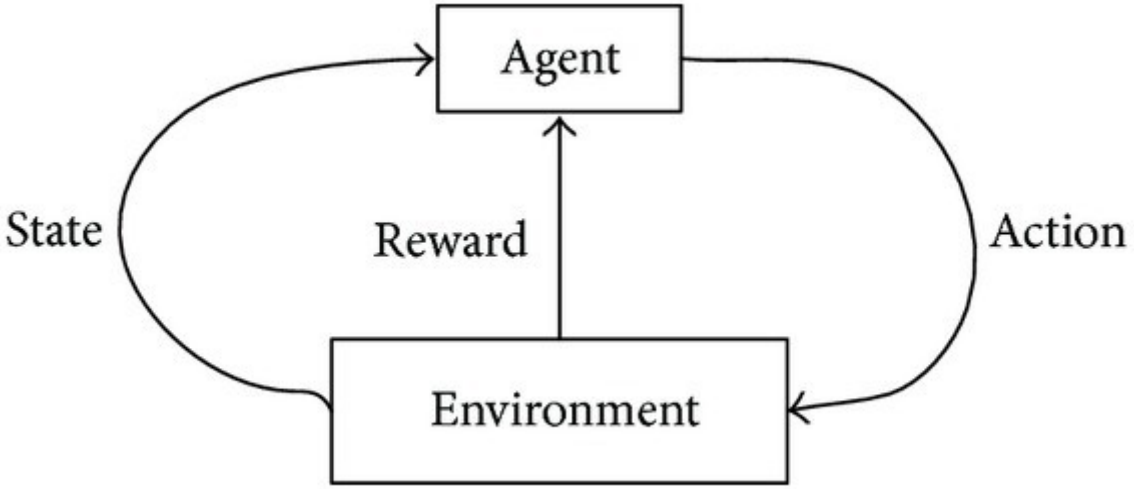
With reinforcement learning you can

- Control robots!
- Personalize medical treatment!
- Optimize advertisement!



- Tackle HEP problems!
- And of course, play games from raw images!
  - Most popular among scientists
- A lot more, ofc

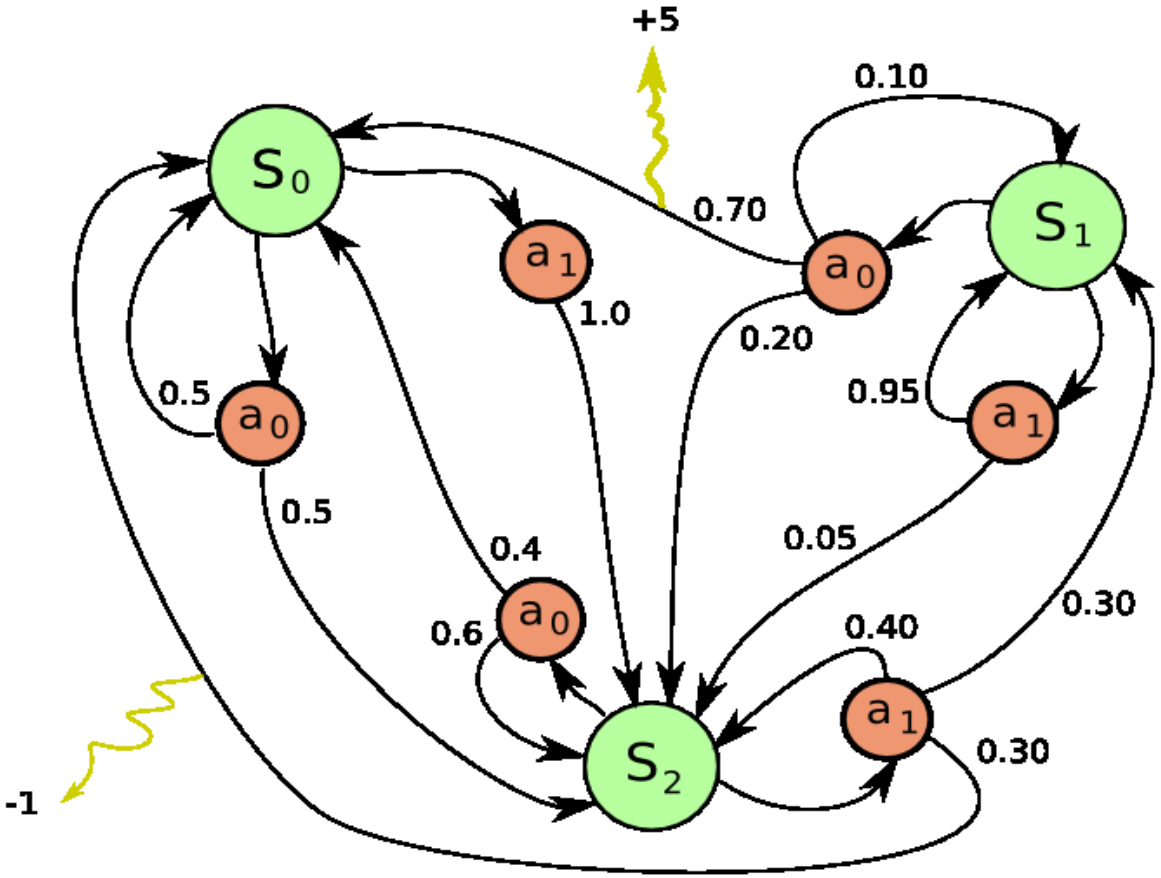
The MDP formalism



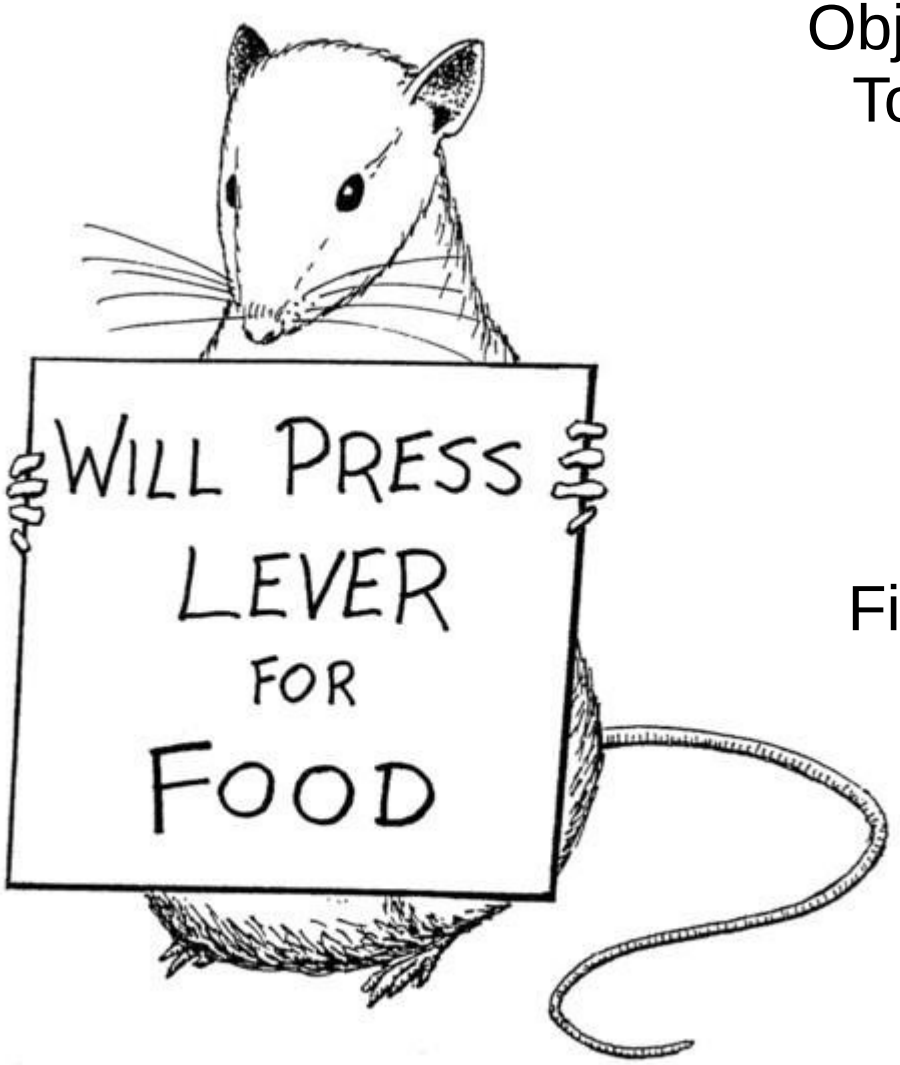
Classic MDP

Agent interacts with environment

- Environment states: $s \in S$
- Agent actions: $a \in A$
- State transition: $P(s_{t+1} | s_t, a_t)$
- Reward: $r_t = r(s_t, a_t)$


+5
0.10
S_0
0.70
a_1
a_0
S_1
1.0
0.20
0.5
a_0
0.95
a_1
0.5
0.05
0.4
a_0
0.6
0.40
0.30
a_1
S_2
0.30
-1

Objective: learn optimal policy



Objective:
Total reward

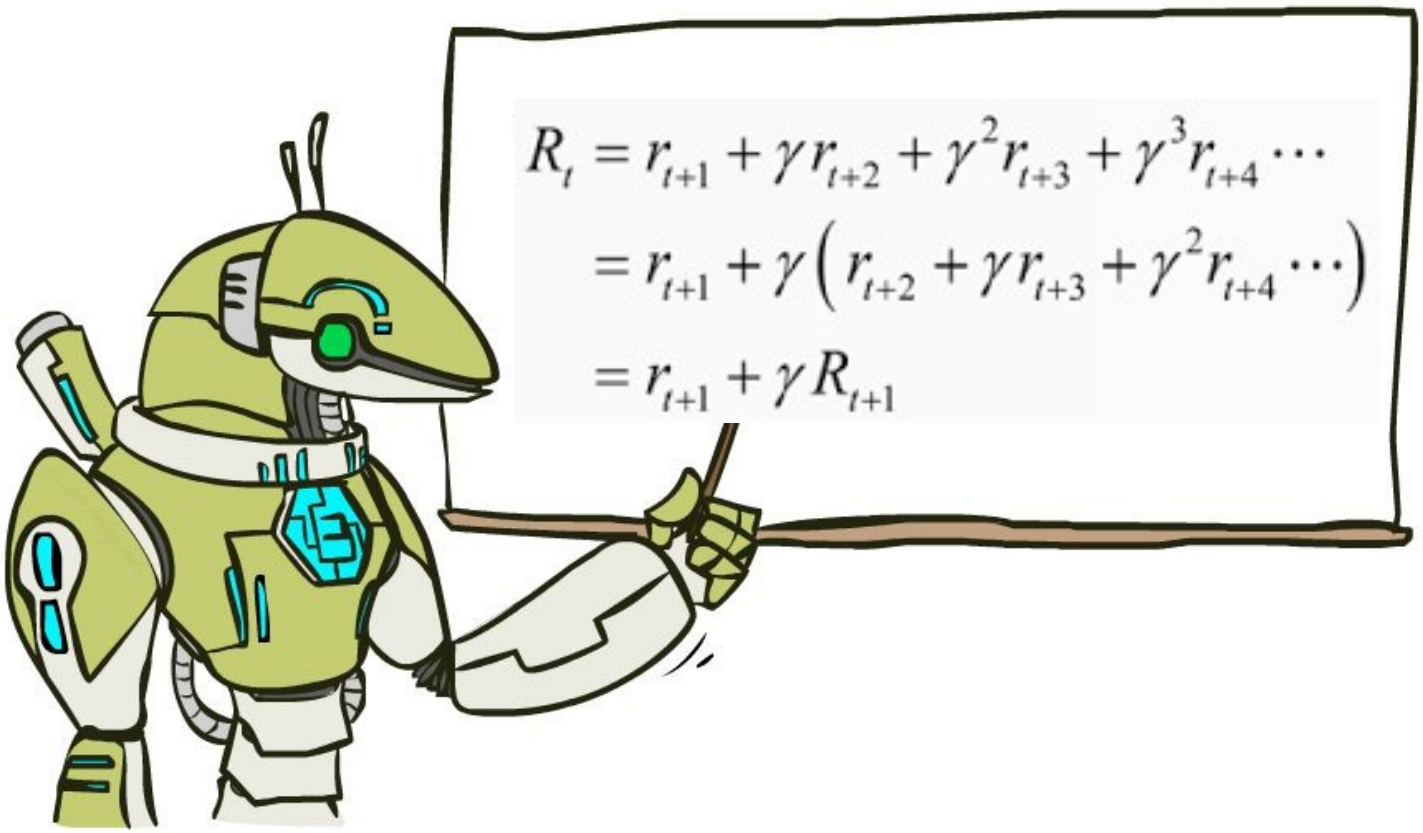
$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots + \gamma^n \cdot r_{t+n}$$

$$R_t = \sum_i \gamma^i \cdot r_{t+i} \qquad \gamma \in (0,1) const$$

Find policy that maximizes total reward

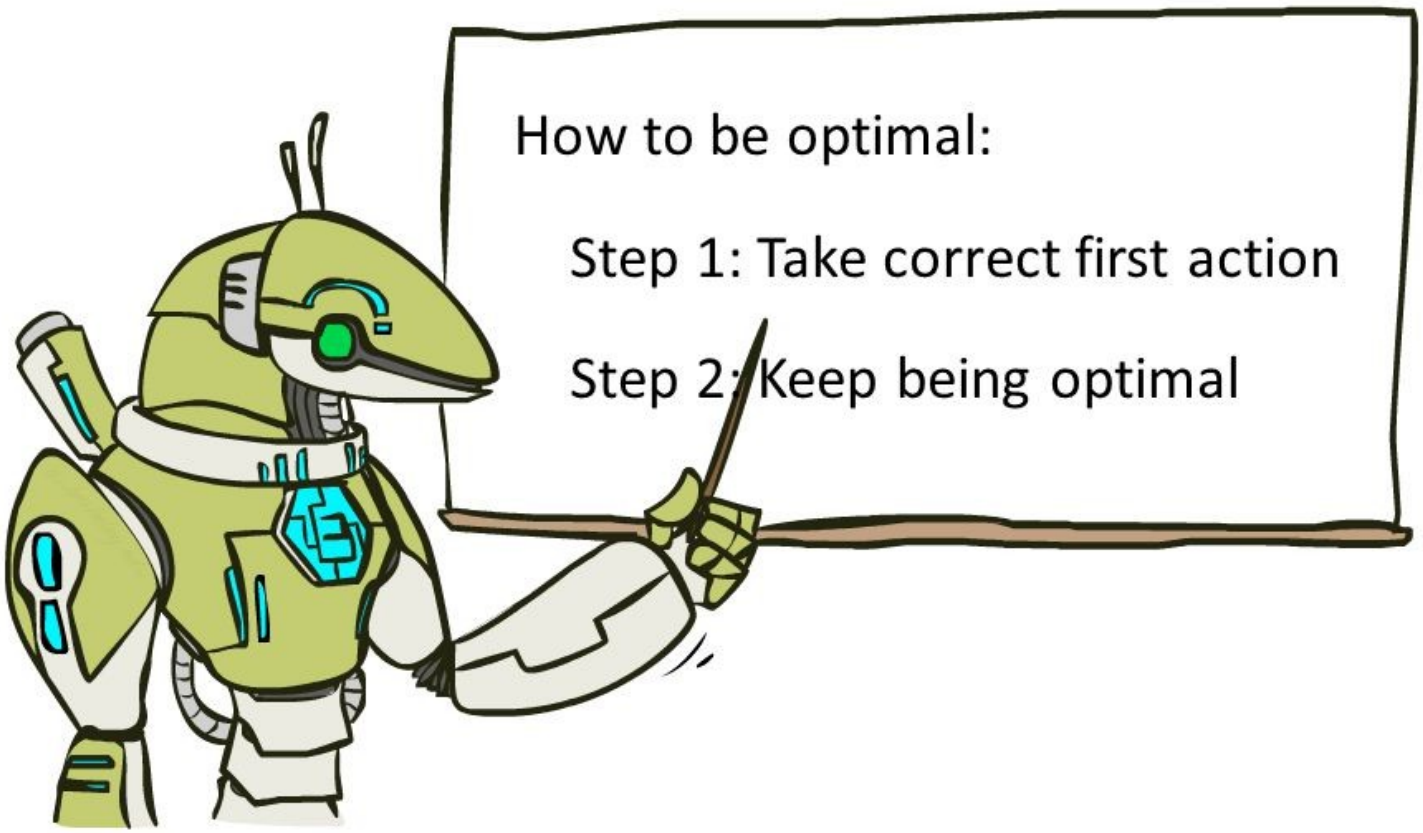
$$\pi = P(a|s) : E[R] \rightarrow max$$

The Bellman Equations



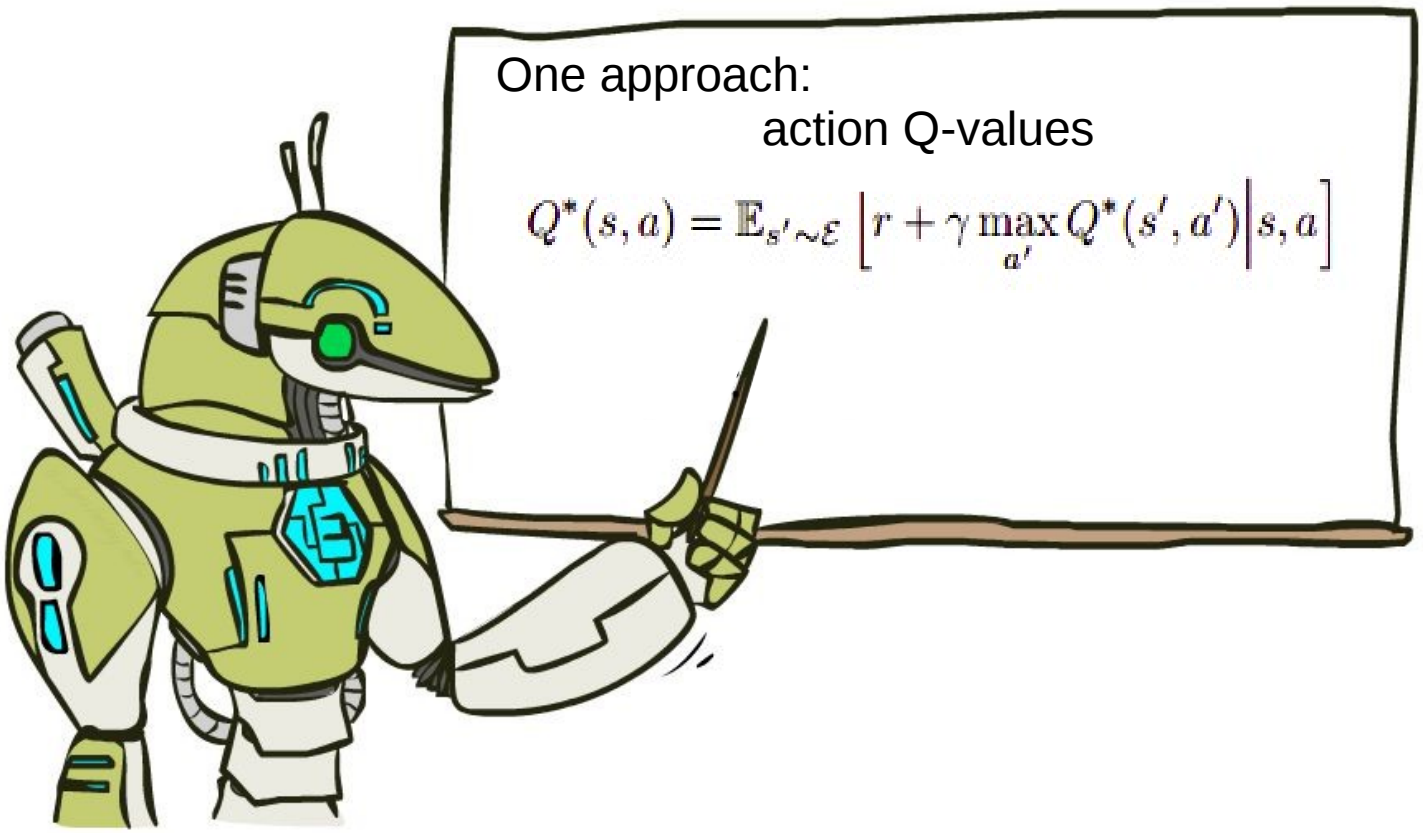
Maximize full session reward

The Bellman Equations



Recurrent optimal strategy definition

The Bellman Equations



$$Q'(s_t,a_t) \leftarrow Q_t(s_t,a_t) - \alpha\left(Q_t^{ref} - Q(s_t,a_t)\right)$$

$$Q_t^{ref} = r_t + \gamma \cdot max_{a'} Q(s_{t+1},a')$$

$$\pi : argmax_a' Q(s,a')$$

Other methods: value-based

$$Q'(s_t, a_t) \leftarrow Q_t(s_t, a_t) - \alpha\left(Q_t^{ref} - Q(s_t, a_t)\right)$$

Q-learning

$$Q_t^{ref} = r_t + \gamma \cdot max_{a'} Q(s_{t+1}, a')$$

SARSA

$$Q_t^{ref} = r_t + \gamma \cdot Q(s_{t+1}, a_{t+1})$$

Qlearning n-step

$$Q_t^{ref} = r_t + \gamma \cdot r_{t+1} + ... + \gamma_{t+n-1} r_{t+n-1} + \gamma_{t+n} \cdot max_{a'} Q(s_{t+n}, a')$$

Other methods: actor-critic

$$\pi(s,a)=P(a|s) \qquad V(s)=E[R_\pi(s)]$$

Advantage actor-critic (a2c)

$$V_\pi(s_t) \leftarrow V(s_t)+\alpha\cdot(r_t+\gamma\cdot V_\pi(s_t+1)-V(s_t))$$

$$\pi(s_t) \leftarrow argmax_\pi \log\pi(s,a)\cdot(r_t+\gamma\cdot V_\pi(s_t+1)-V(s_t))$$

Idea:

- take action more frequently if it exceeds expected value.
- abandon actions that fall below expected value

Other methods: actor-critic

$$\pi(s,a)=P(a|s) \qquad V(s)=E[R_\pi(s)]$$

Advantage actor-critic (a2c)

$$V_\pi(s_t) \leftarrow V(s_t)+\alpha\cdot(r_t+\gamma\cdot V_\pi(s_t+1)-V(s_t))$$

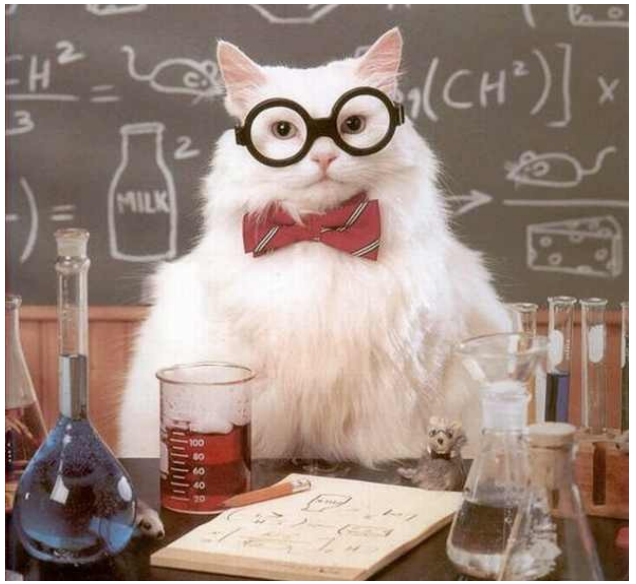
$$\pi(s_t) \leftarrow argmax_\pi \log\pi(s,a)\cdot(r_t+\gamma\cdot V_\pi(s_t+1)-V(s_t))$$

Idea:

- take action more frequently if it exceeds expected value.
- abandon actions that fall below expected value

Exploration Vs Exploitation

Balance between using what you learned and trying to find something even better



Vs

Exploration Vs Exploitation

Strategies:

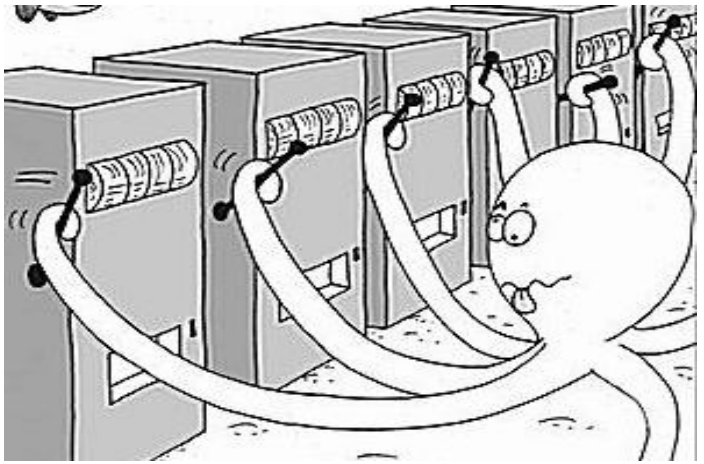
- ε-greedy
  - With probability ε take a uniformly random action; otherwise take optimal action.
- Softmax

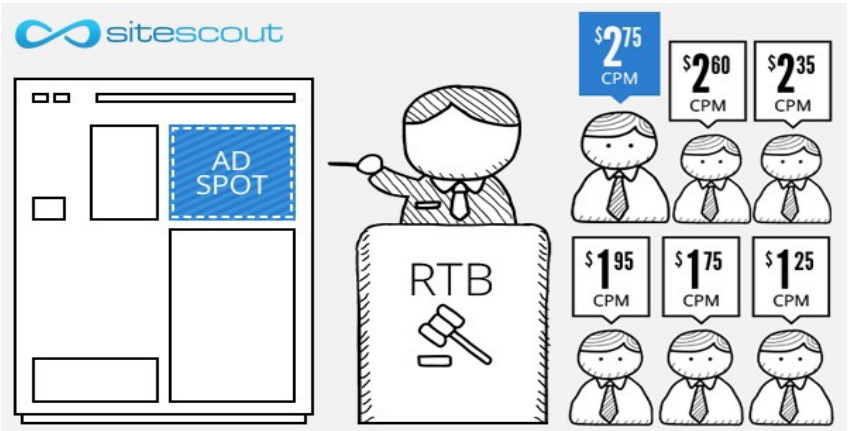
  Pick action proportional to softmax of shifted normalized Q-values.

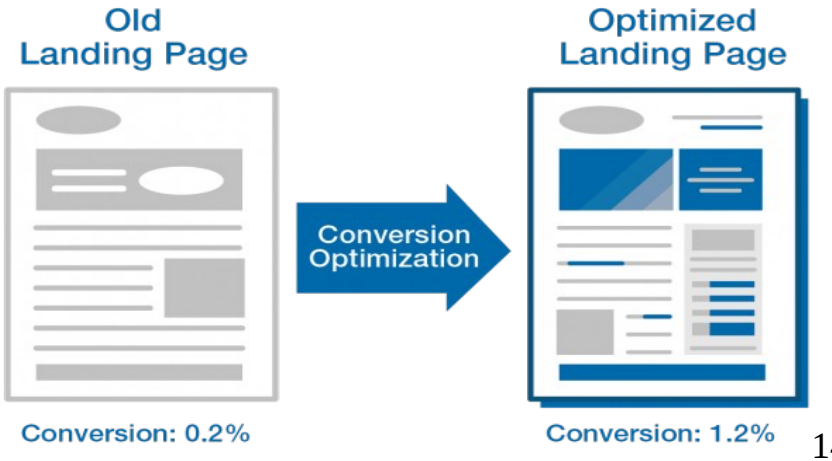
$$P(a) = softmax\left(\frac{Q(a) - Qmean}{Qvariance}\right)$$

- Some methods have a built-in exploration strategy (e.g. A2c)

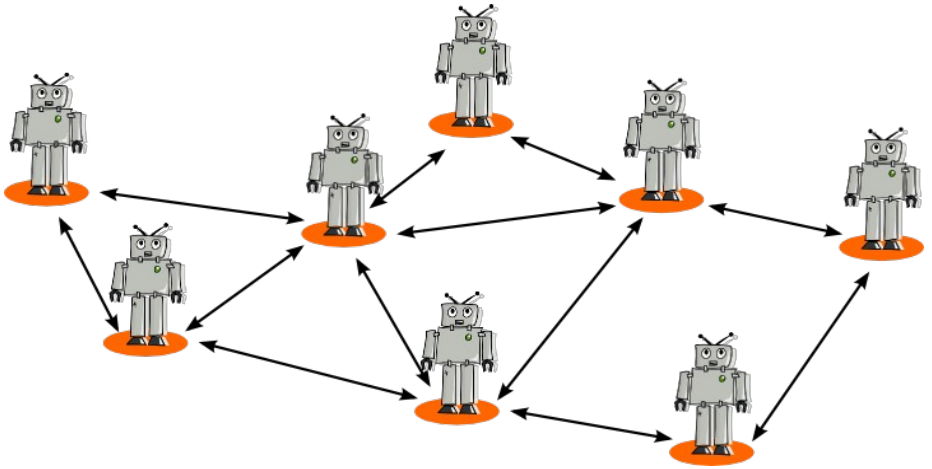
Use cases: MAB

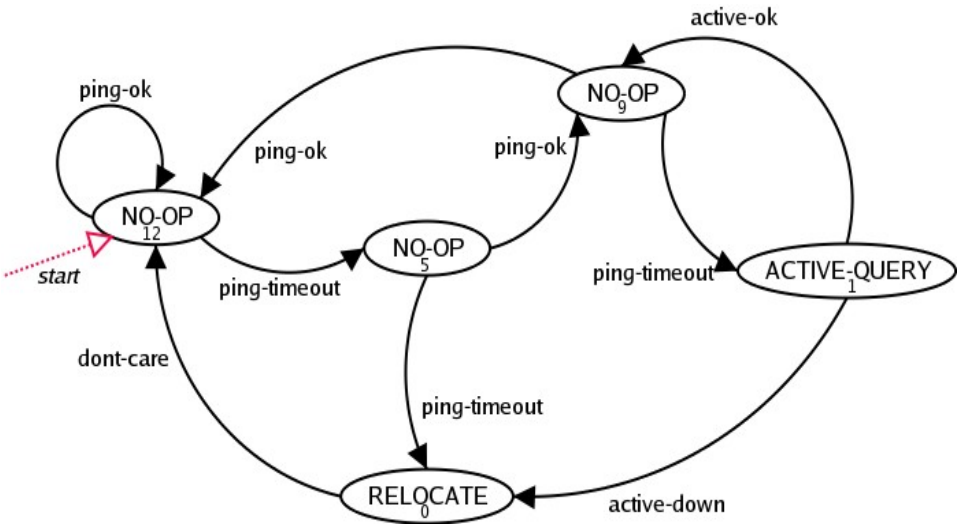




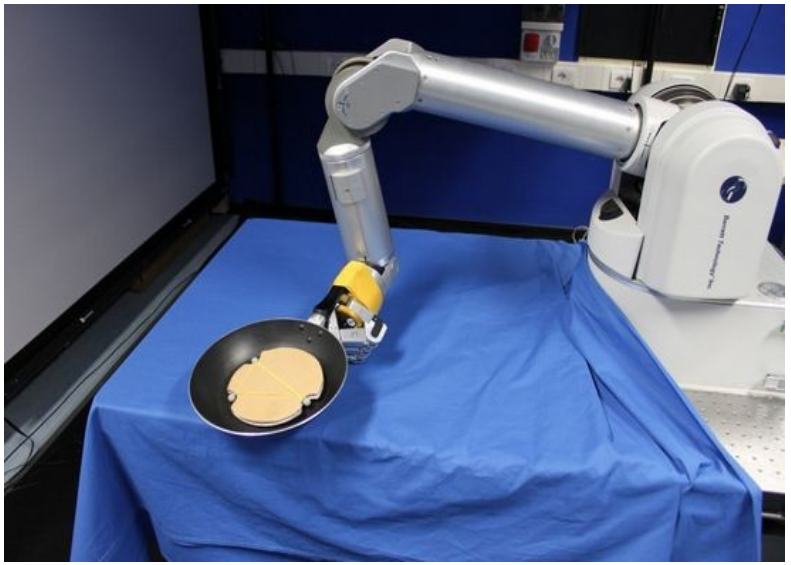


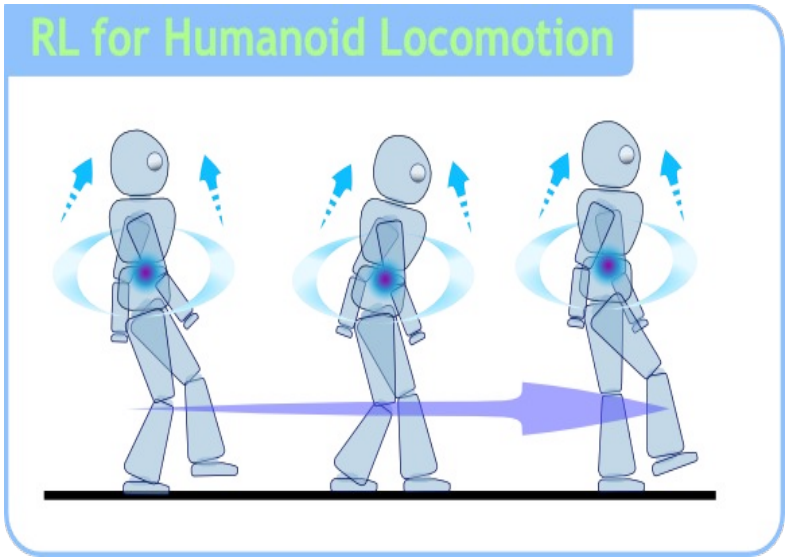
Use cases: multi-agent MDP



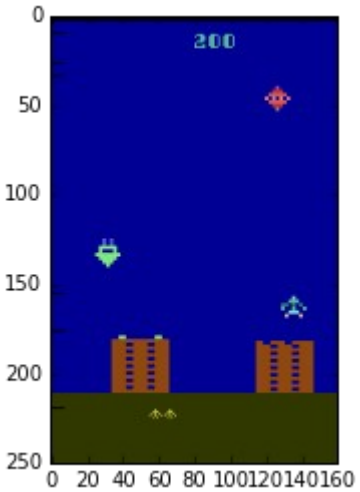


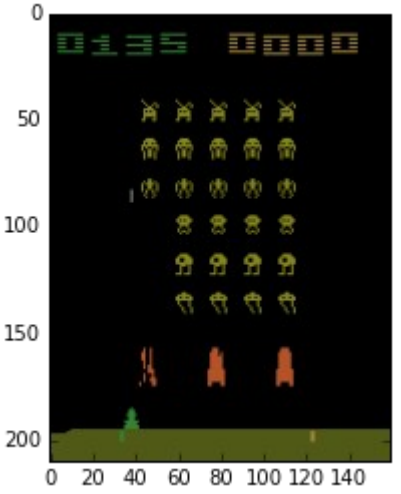
Use cases: dynamic systems

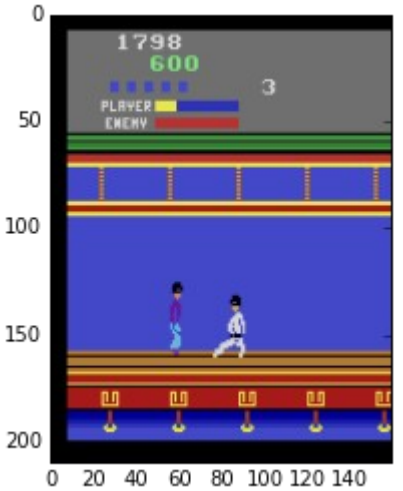




Use cases: videogames



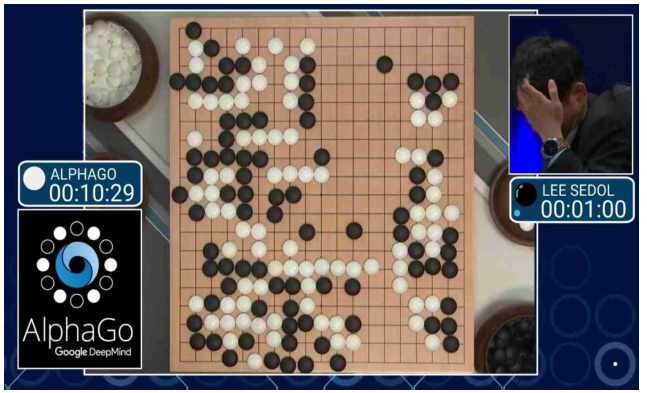




Other use cases

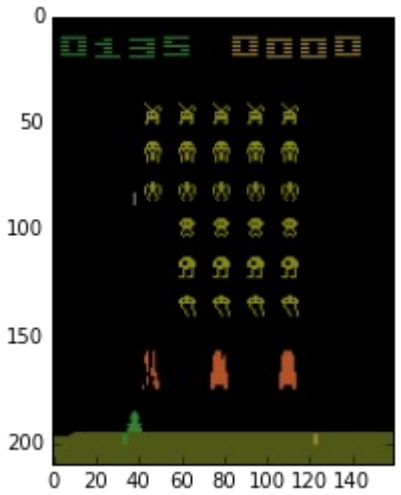
- Personalized medical treatment
  (afaik, prototypes only) http://bit.ly/1VDejIU

- Even more games (Go, chess, etc)



- Conversation systems (next time maybe)

- Could save lives
- Could make money
- Could do robotics
- Chose to play games



Problem:

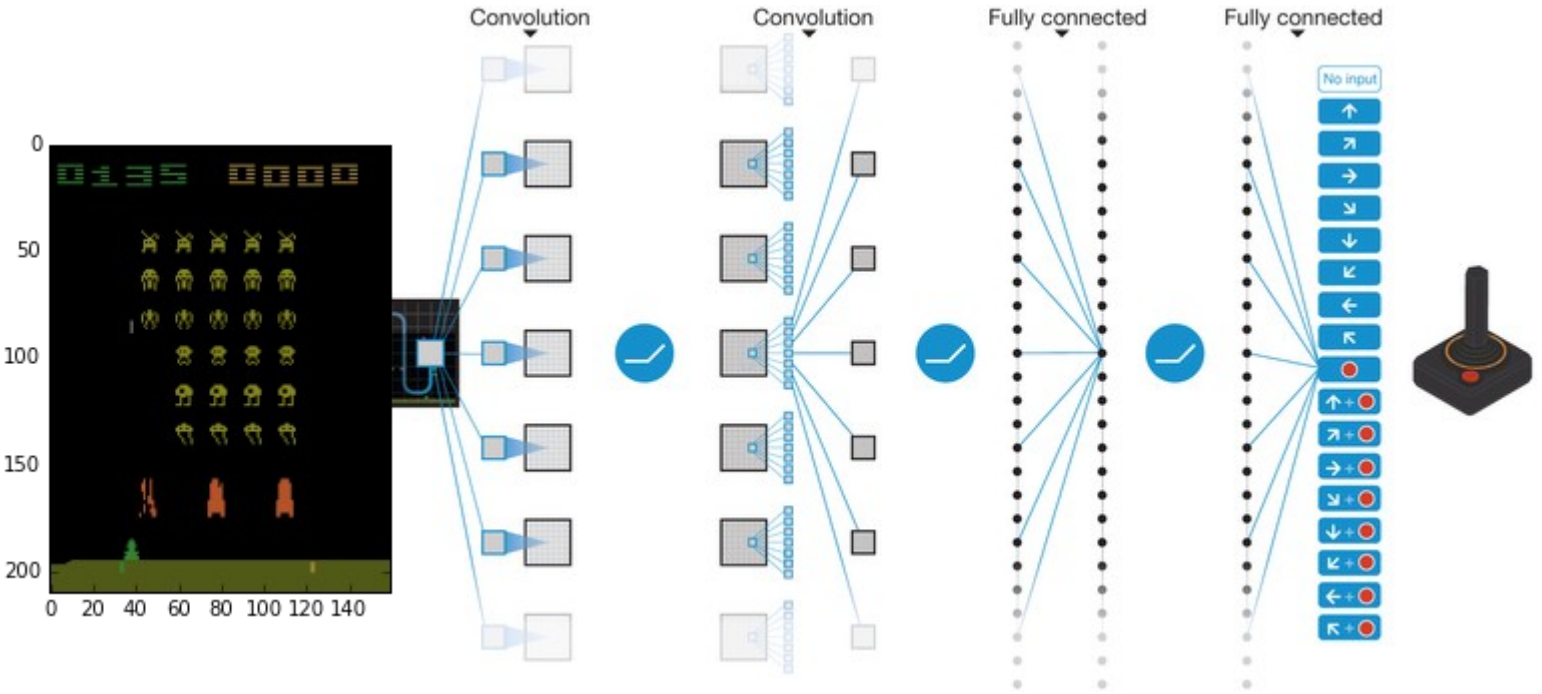
State space is usually large,

sometimes continuous.

And so is action space;

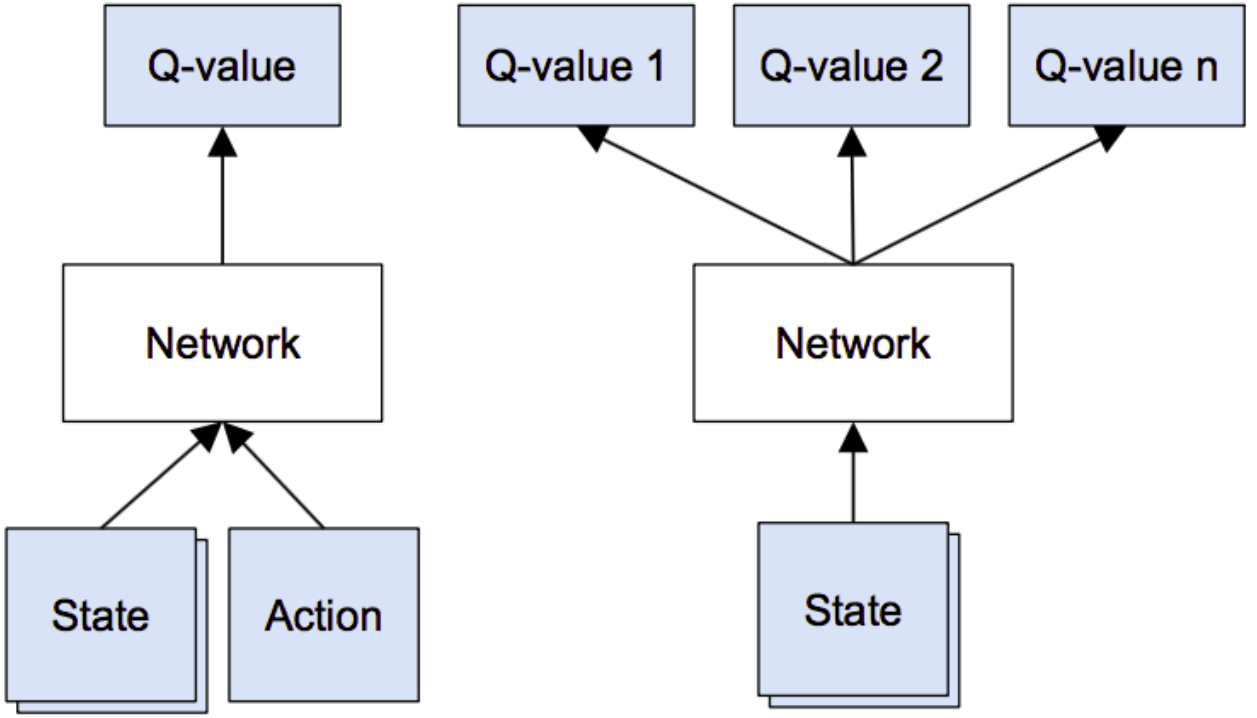
However, states do have a structure, similar states have similar action outcomes.

Deep learning approach: DQN



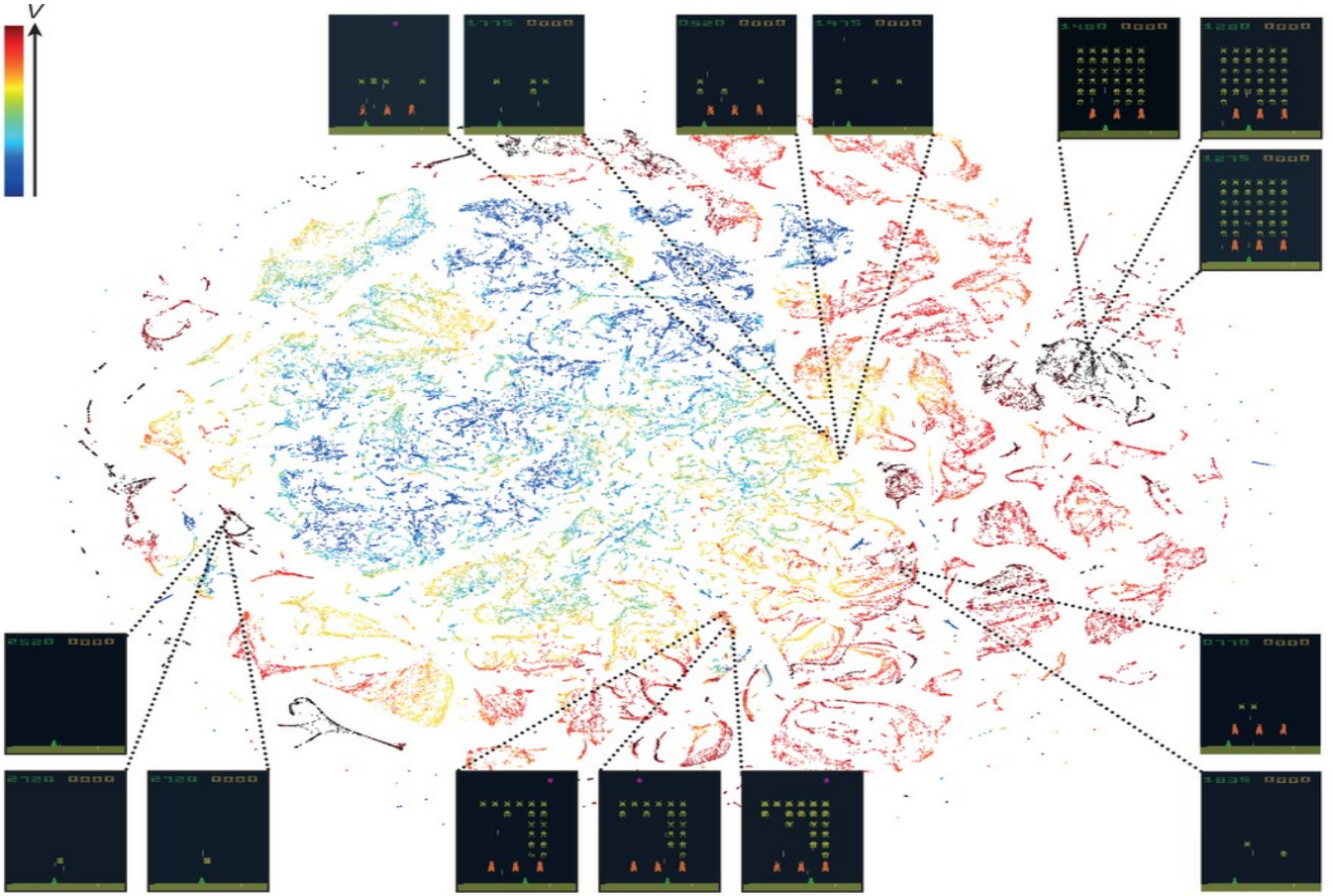
Mnih et al.http://bit.ly/25DqgkH

Deep learning approach: DQN

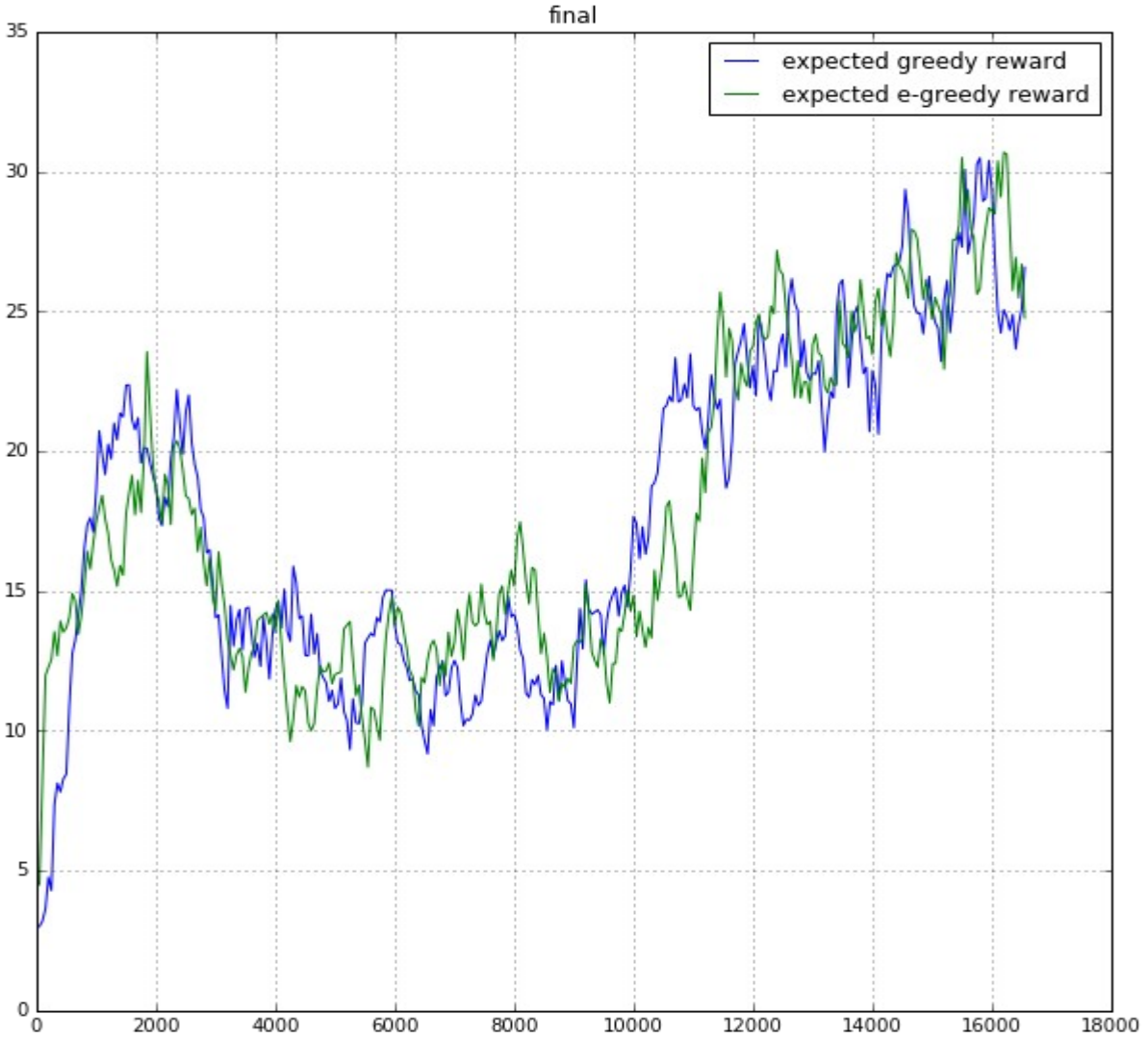


$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

Deep learning approach: DQN



DQN problem: autocorrelations



DQN problem: autocorrelations

Experience replay

- Maintain a large pool of (s,a,r,s') tuples from prior MDP sessions.
- Sample random batch from the pool each time when training NN
  - Or use a prioritized sampling strategy to emphasize most important samples

Silver et al. http://arxiv.org/abs/1511.05952

Target networks

- Obtain “Qreference(s,a)” term from an older neural network snapshot.
  - Alternatively, maintain an exponential moving average of weights

Silver et al. http://jmlr.org/proceedings/papers/v32/silver14.pdf

DQN problem: autocorrelations

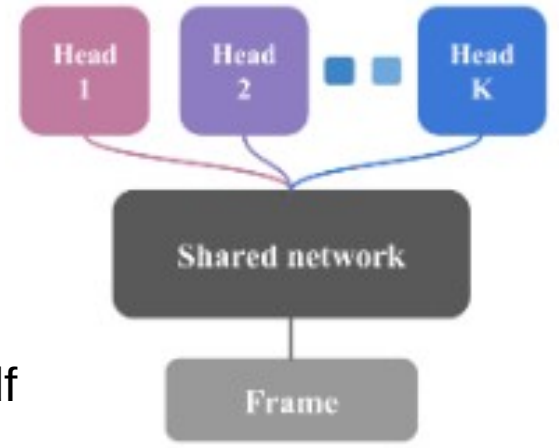
Asynchronous methods

- Launch several parallel agents with shared weights in hope that they will be exploring different environment parts.

Mnih et al. https://arxiv.org/pdf/1602.01783.pdf

Bootstrap DQN

- Maintain several “heads”, top layers of NN responsible for Qvalues prediction.
- At the beginning of new game session, choose one of the “heads” at random. This head decides what action to take during current session.
- All other heads are trained on that session without taking real actions

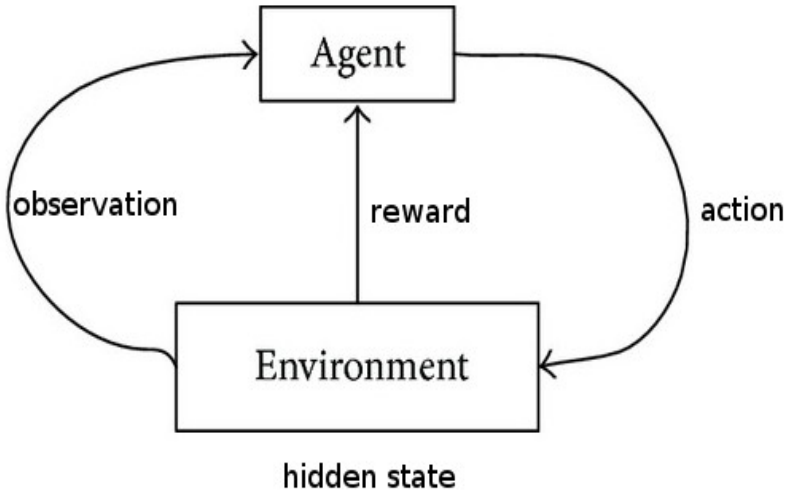


Van Roy et al. https://arxiv.org/pdf/1602.04621.pdf

Problem:

Most practical cases are partially observable:

Agent observation does not hold all information about process state (e.g. human field of view).



- We now receive observations instead of actual environment states

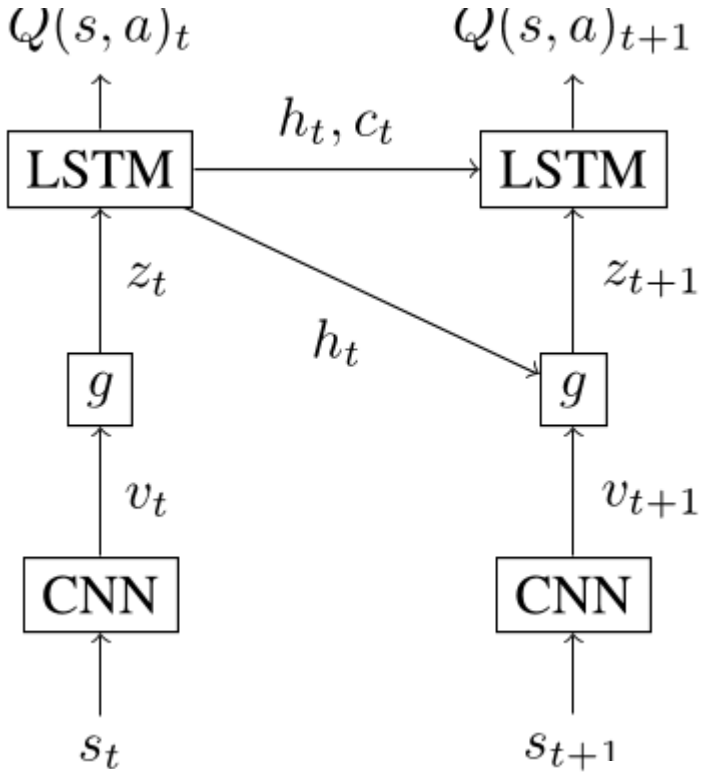
$$o \in O; O \neq S$$

- However, we can try to infer hidden states from sequences of observations.

$$s_t \simeq m_t : P\left(m_t | o_t, m_{t-1}\right)$$

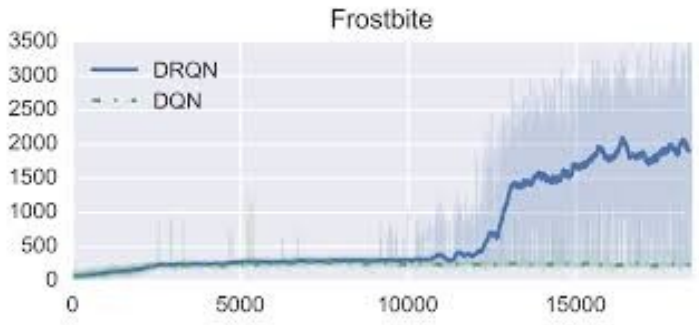
- Intuitively that's agent memory state.

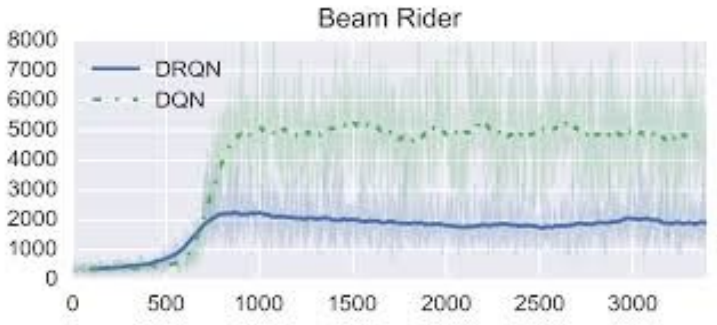
Deep Recurrent RL

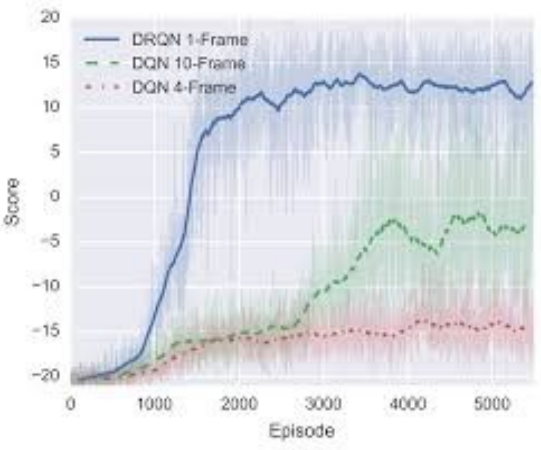


Recurrent agent memory

- Agent has his own hidden state.
- Trained via BPTT with a fixed depth
- Problem: next input depends on chosen action
- Even more autocorrelations :)



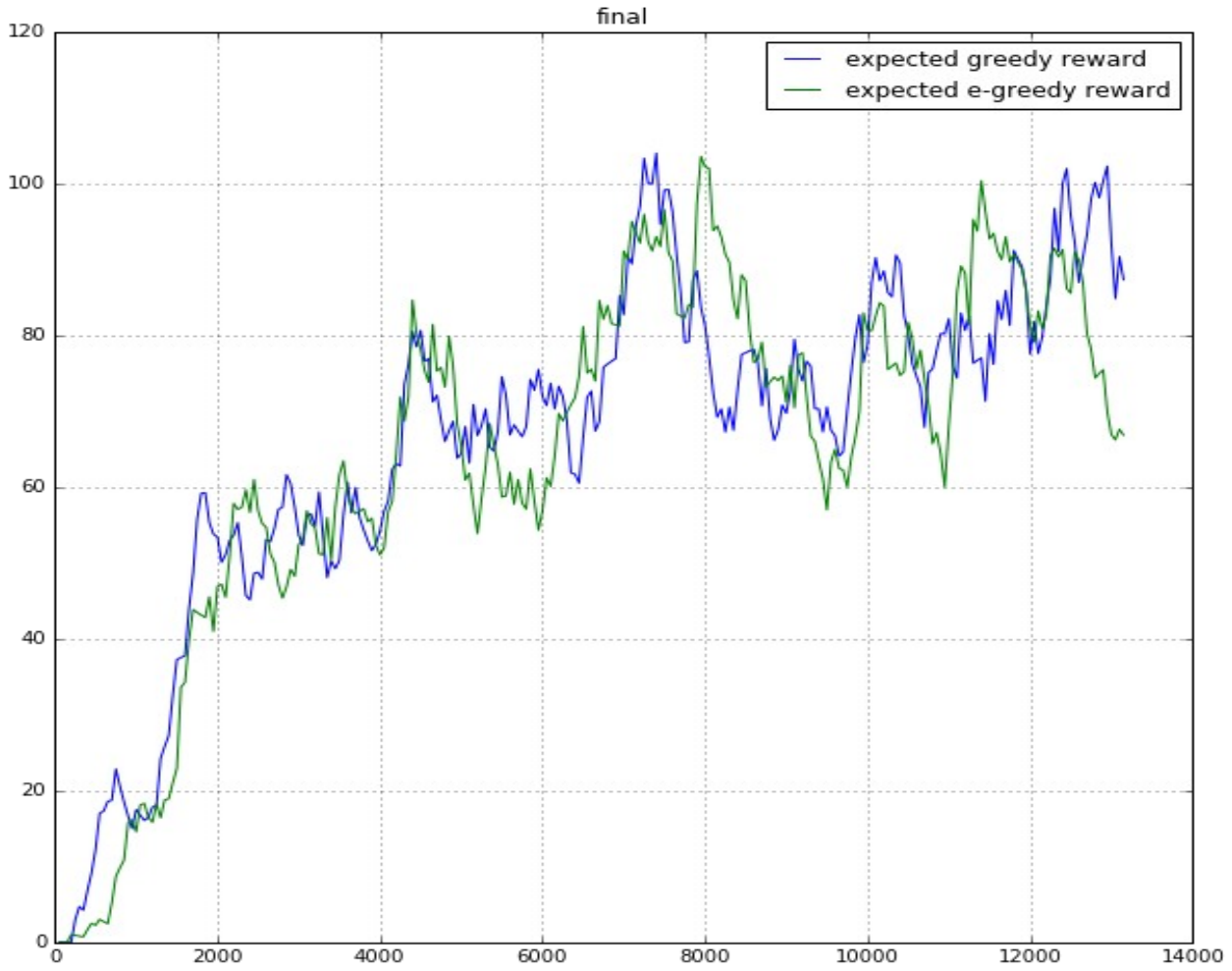




http://arxiv.org/pdf/1507.06527.pdf http://arxiv.org/abs/1507.06527

Deep Recurrent RL

Learning curves for KungFuMaster



Problem:

Rewards are usually sparse (temporally rare) and delayed.

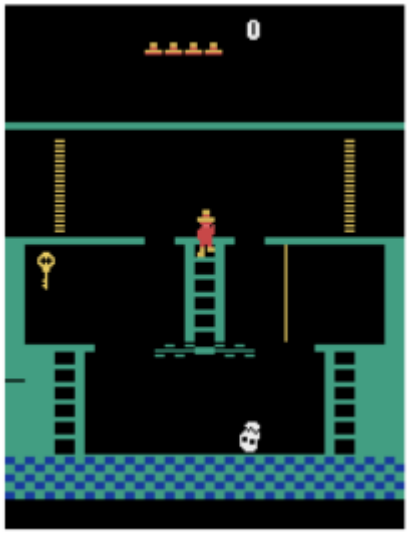
It takes exponentially more random exploration to learn optimal policy in case of rare rewards.

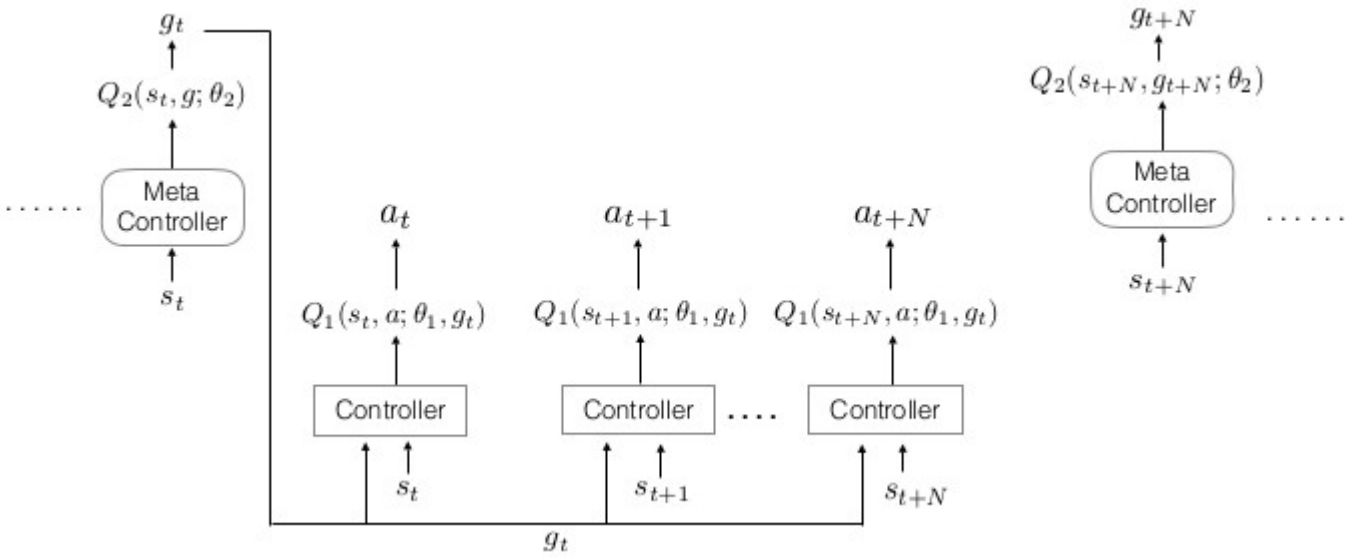
Spoiler, in real life things are even more sparse and delayed

Humans:

- Don't seem to follow epsilon-greedy exploration policy (e.g. random limb tremble)
- Think in several layers of abstraction
  - “Push gas pedal (while driving)”
  - “Take left turn in 15 meters”
  - “Drive to school”,
  - “Give your children education”
  - …
  - PROFIT

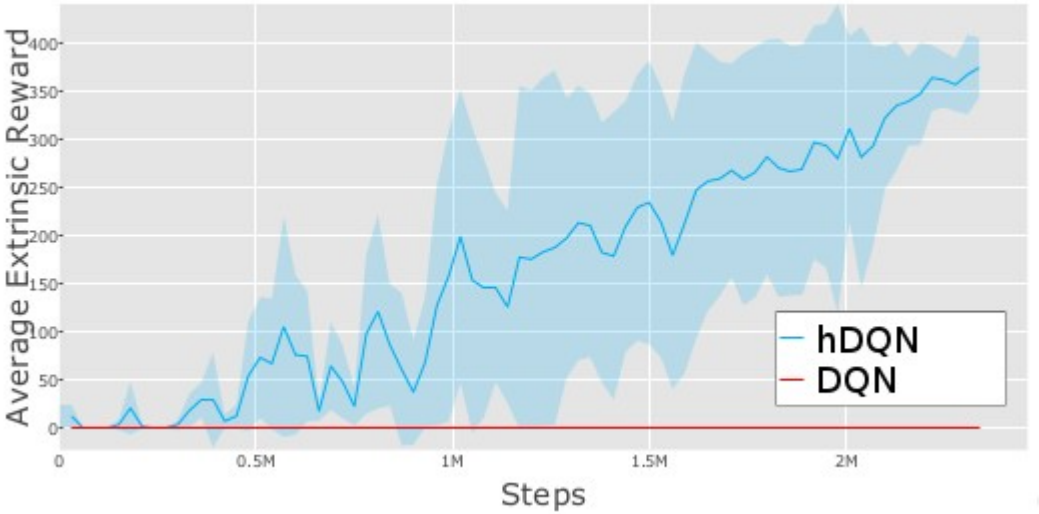
Hierarchical deep RL





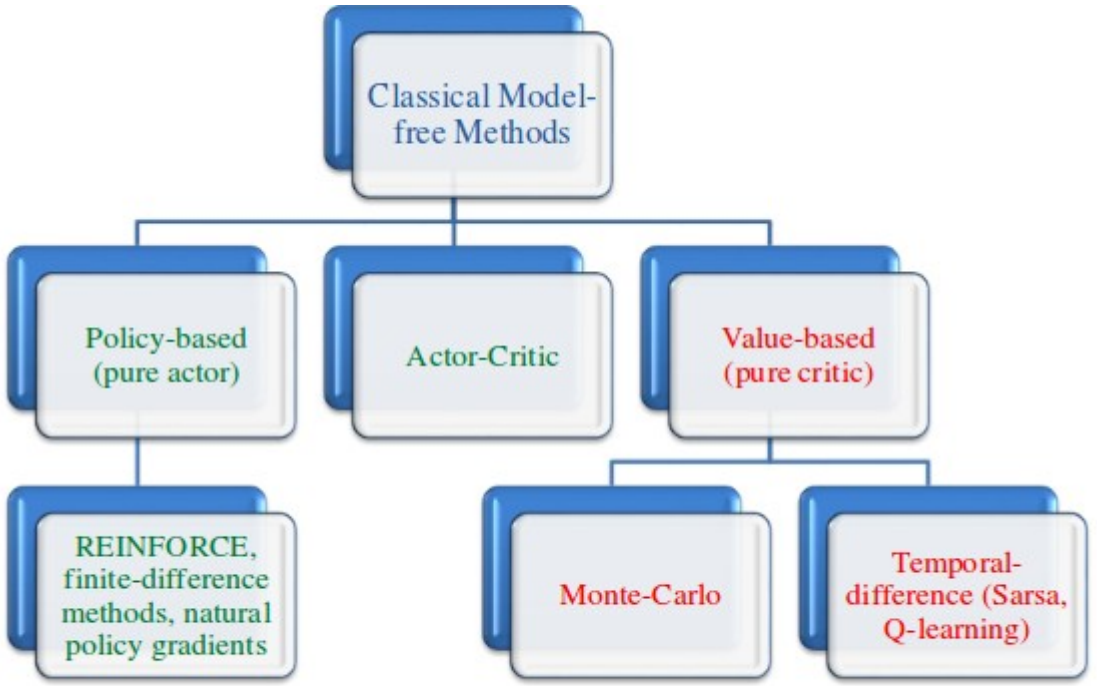
Tenenbaum et al. https://arxiv.org/abs/1604.06057v1

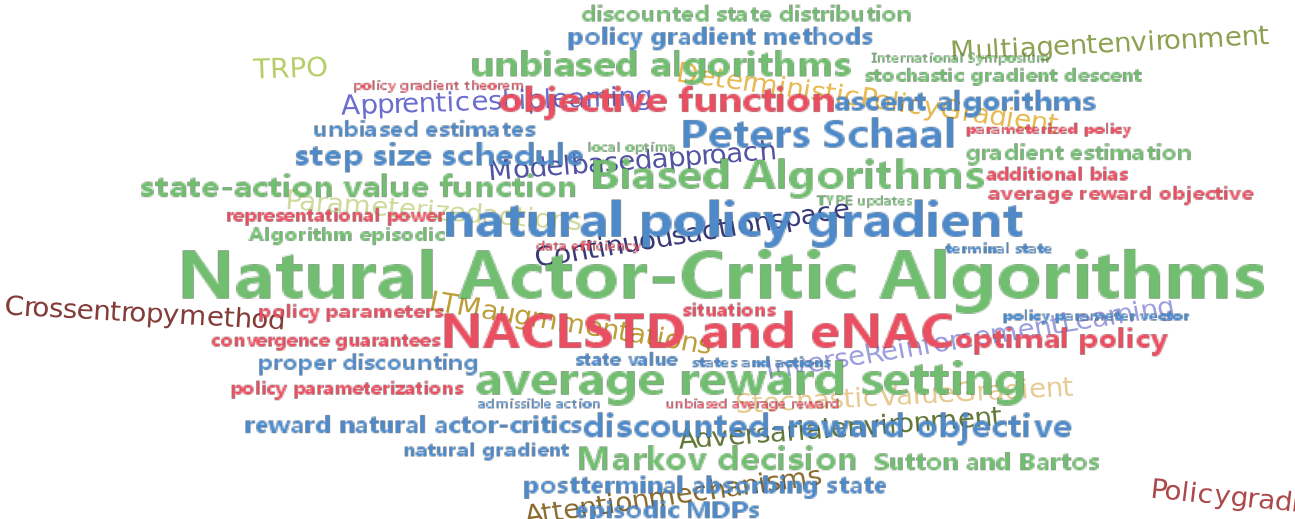
Hierarchical deep RL



Tenenbaum et al. https://arxiv.org/abs/1604.06057v1

Expected rabbit hole depth





Most important slide

RL isn't magical

- It won't learn everything in the world given any data
  and random architecture.

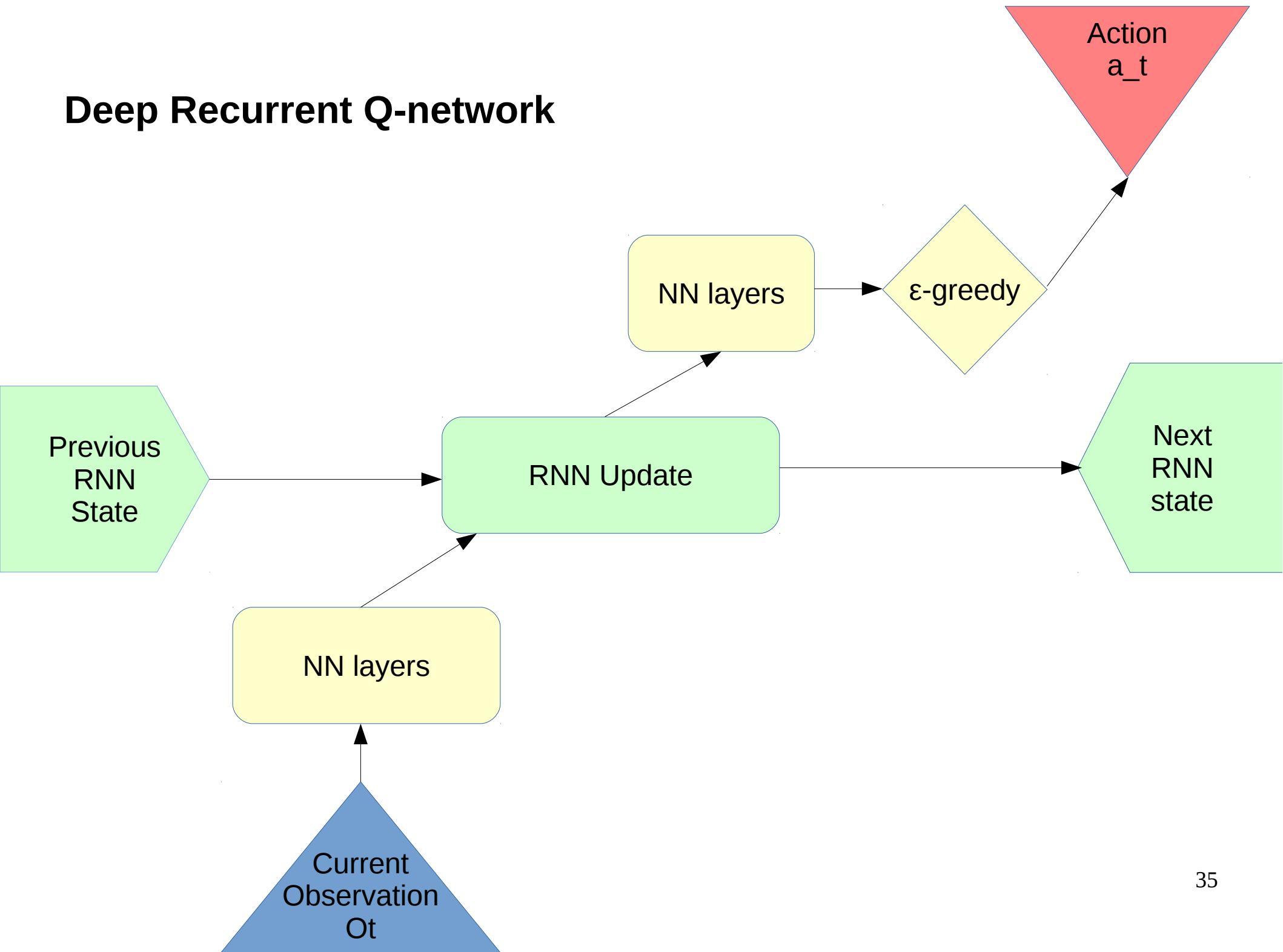
- Sparse & delayed rewards still a problem

- Less playing Atari, more real world problems
  No, doom is not a real world problem, dummy!

- Slowly getting rid of heuristics towards mathematical and neurological soundness

- High entry threshold
  A lot of technical issues to solve;
  Environment simulation or hardware;
  Deep learning libraries;

Deep Recurrent Q-network



AgentNet

In [2]:
```
#agent observation input (blue triangle)
observation_layer = InputLayer(observation_size,name="obs_input")

#previous state input (left green)
prev_state_layer = InputLayer(rnn_size,name="prev_state_input")

#new RNN state (middle and right green)
rnn = RNNCell(prev_state,
              observation,
              name="new rnn state")

#Qvalues estimator (red square)
q_values = DenseLayer(rnn, name="QEvaluator",
                      num_units = n_actions,
                      nonlinearity=None,)

#Epsilon-greedy action picker (red triangle)
resolver = EpsilonGreedyResolver(q_values,name="resolver")

#all together
agent = Agent(observation_layer,
              {rnn:prev_state_layer},
              q_values,resolver)
```

AgentNet

https://github.com/yandexdataschool/AgentNet

Basically a library to quickly experiment with different agent compositions and learning methods.

- MDP abstractions: agent, environment, etc
- Various recurrent and LTM augmentations
- Several RL algorithms:
  - SARSA, Q-l, K-step Ql, Advantage A2c, …

- Full lasagne compatibility

- User defines one agent step
  - Processing observations
  - Taking actions
  - Updating memory states

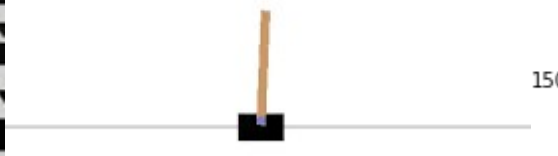
Lasagne

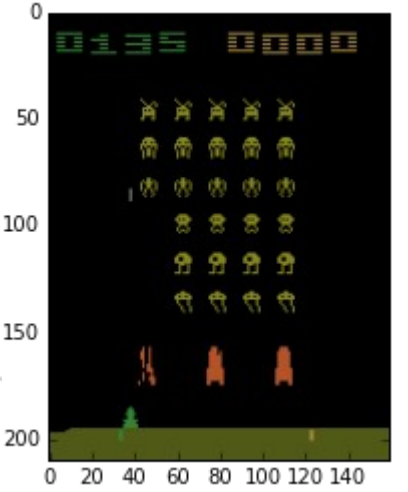
- Minimalistic DL library
- Built above Theano
- Direct theano integration
- Great docs, recipes

OpenAI Gym

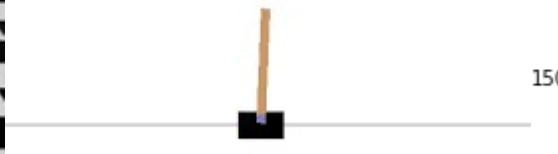
- Opensource RL problem zoo
- Atari, MujoCo, classic games, etc
- Gone open-source

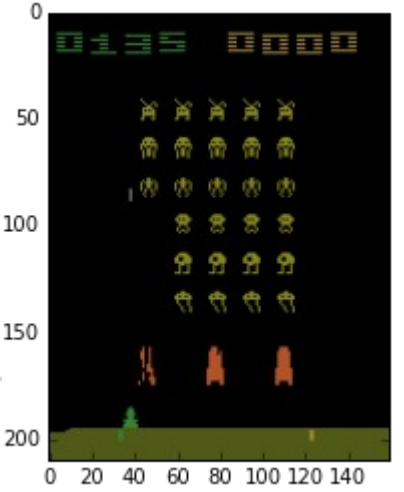
OpenAI





Contributors **VERY** welcome





</speech>

- Any questions?
- Ideas, feedback and contributions are very welcome.
- https://github.com/yandexdataschool/AgentNet
- See ./examples for... examples