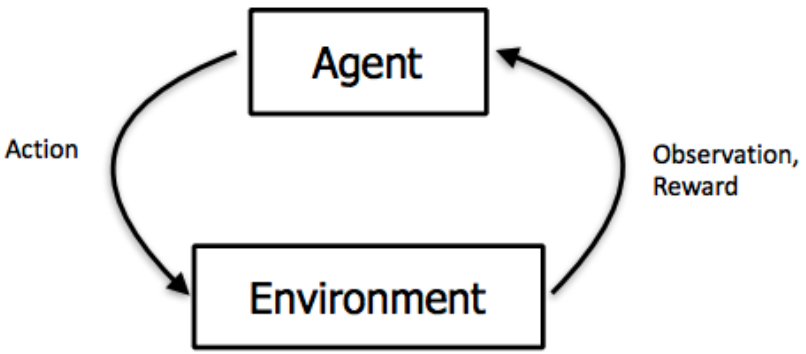
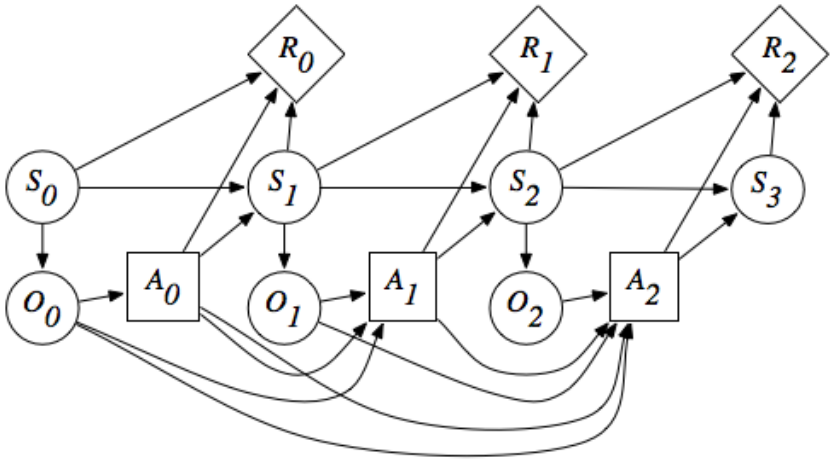
Gentle introduction to Deep Reinforcement Learning

06.06.2019

Reinforcement learning





Let $s \in S$ denote *state*, $a \in A$ – *action*, τ – *trajectory*, θ – model parameters, π – *policy*, $p(s_{t+1}|s_t, a_t)$ – transition operator.
The objective of reinforcement learning:

$$\pi_\theta(\tau) = p_\theta(s_1, a_1, ..., s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

$$\theta^* = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

Value based methods (tabular case)

$$V^{\pi}(s) = \mathrm{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} | S_t = s\right], Q^{\pi}(s,a) = \mathrm{E}_{\pi}\left[\sum_{k=t}^{\infty}\gamma^{k} r_{t+k+1} | S_t = s, A_t = a\right]$$

$$\begin{aligned} V^{\pi} &= \sum_{a}\pi(a|s)\sum_{s',r} p(s',r|s,a)\left[r_t + \gamma V_{\pi}(s')\right] \\ Q^{\pi} &= \sum_{s',r} p(s',r|s,a)\left[r + \gamma\sum_{a'}\pi(a'|s')Q_{\pi}(s',a')\right] \end{aligned}$$

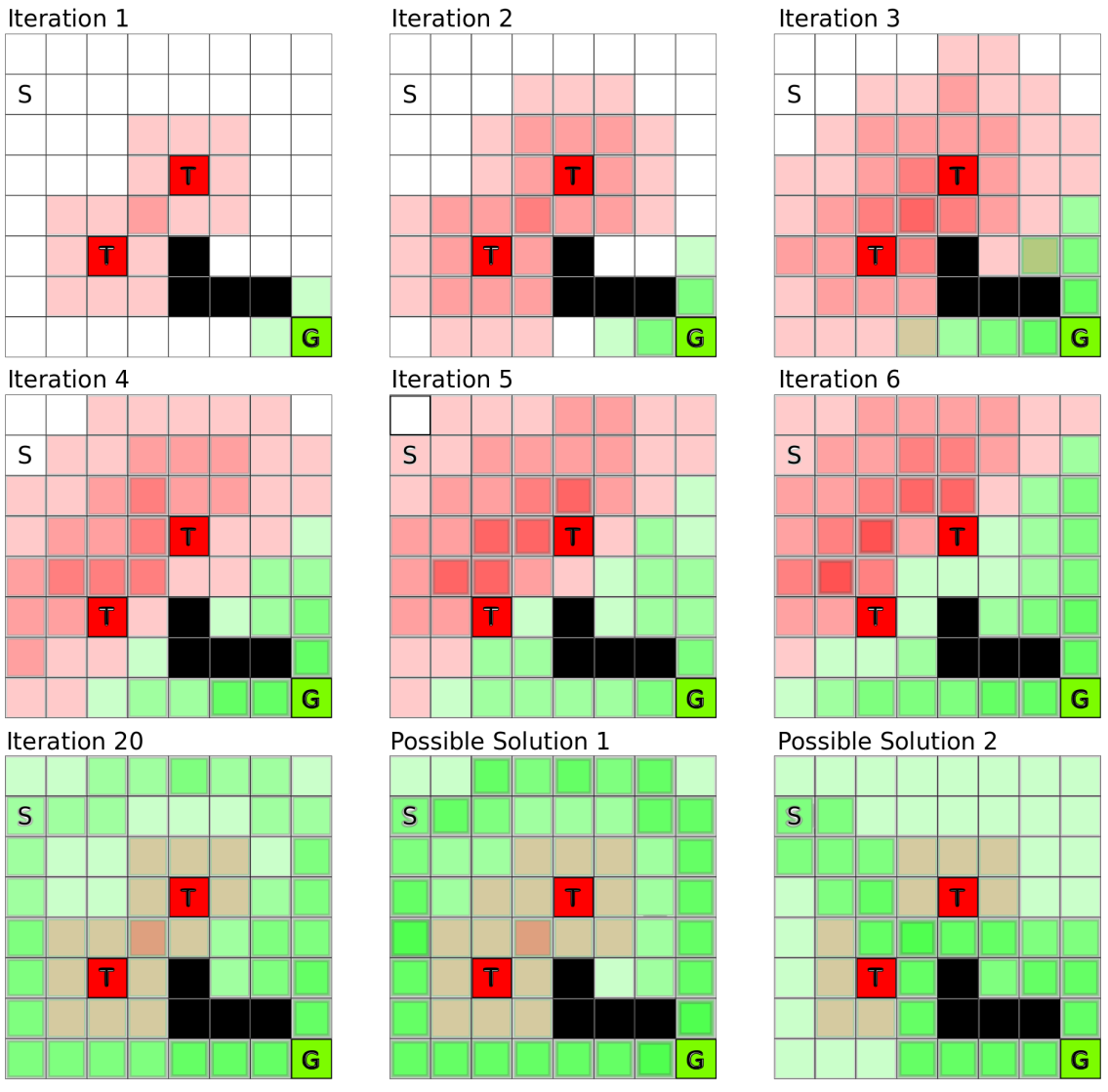
Policy greedy to Q: $\pi'(a_t|s_t) = \begin{cases} 1 \text{ if } a_t = \arg\max_{a_t} Q(s_t,a_t), \\ 0 \text{ otherwise} \end{cases}$

Bellman equation: $Q(s,a) \leftarrow r(s,a) + \gamma \mathrm{E}\left[V(s')\right]$

Steps at each iteration of Value Iteration algorithm:

1. $Q(s,a) \leftarrow r(s,a) + \gamma \mathrm{E}_{s' \sim p(s'|s,a)}\left[V(s')\right]$
2. $V(s) \leftarrow \max_a Q(s,a)$ (the same as $\pi \leftarrow \pi'$)

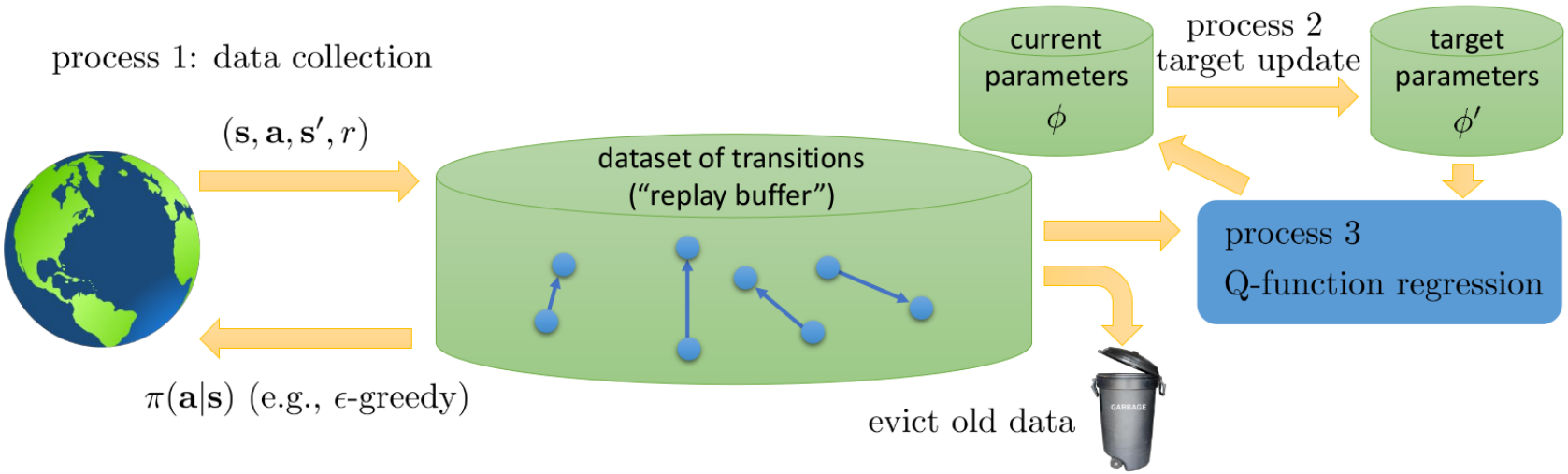
Value based methods (tabular case)



Value based methods: classic DQN

---

for $i = 0, i < N, i++$ **do**
  take a step in the environment using some policy and add the transition (s_i, a_i, s'_i, r_i) to experience replay buffer $\mathcal{B}$
  if $i\%\nu_{target} = 0$ **then**
    update target network parameters: $\phi' \leftarrow \phi$
  if $i\%\nu_{train} = 0$ **then**
    sample a batch of size b $\{(s_j, a_j, s'_j, r_j)\}$ from $\mathcal{B}$
    $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(s_j, a_j)\,(Q_\phi(s_i, a_i) - [r_i + \gamma \max_{a'} Q_{\phi'}(s', a')])$



Policy gradient methods

Idea: differentiate the reinforcement learning objective:

$$J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right], \theta^* = \arg\max_\theta J(\theta)$$

$$J(\theta) = \int \pi_\theta(\tau) r(\tau) d\tau, \nabla J(\theta) = \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau$$

Log derivative trick

$$\pi(\tau) \nabla_\theta \log \pi_\theta(\tau) = \pi_\theta(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \nabla_\theta \pi_\theta(\tau)$$

$$\nabla_\theta J(\theta) = \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta r(\tau) d\tau = E_{\tau \sim \pi_\tau} \left[\nabla_\theta \log \pi_\theta(\tau) r(\tau) \right]$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \right) \left(\sum_{t=0}^{T} r(s_t^i, a_t^i) \right)$$

Policy gradient methods

Algorithm 1 REINFORCE

for $i = 0, i < K, i++$ **do**
sample trajectories $\{\tau^i\}_{i=1}^N$ from $\pi_\theta(a_t|s_t)$ (run the policy)
$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\right) \left(\sum_t r(s_t^i, a_t^i)\right)$
$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Modern modification – Proximal Policy Optimization (Openai 2017). Does not allow agent to change policy to much on each iteration.

Actor-critic

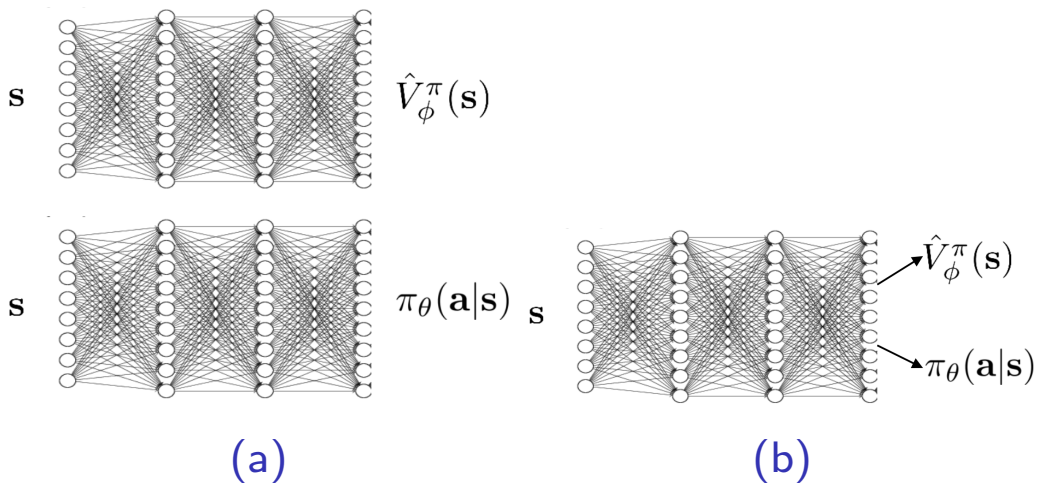
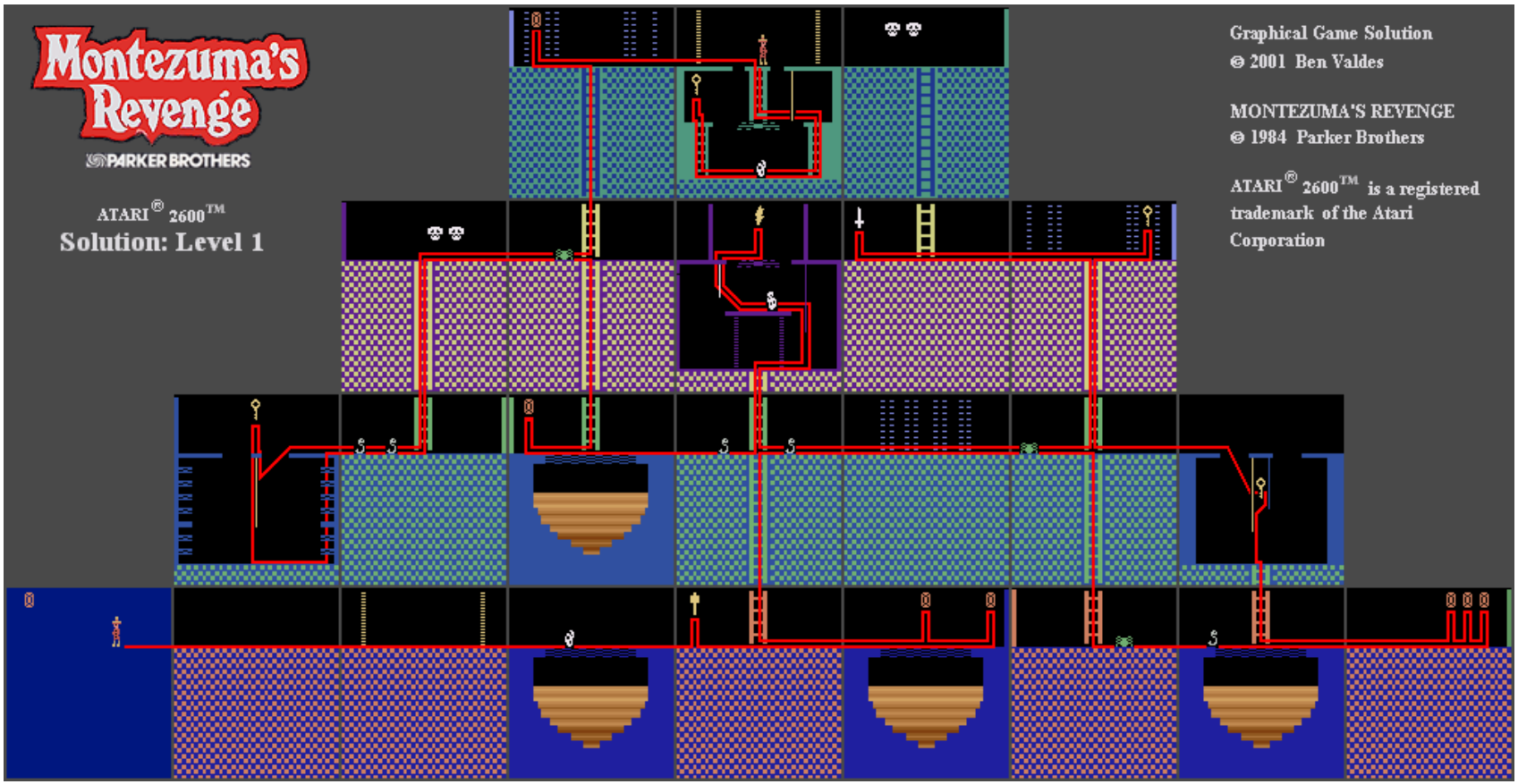


Figure 1: Network architectures for actor-critic method (a) Two networks design (b) Shared network design

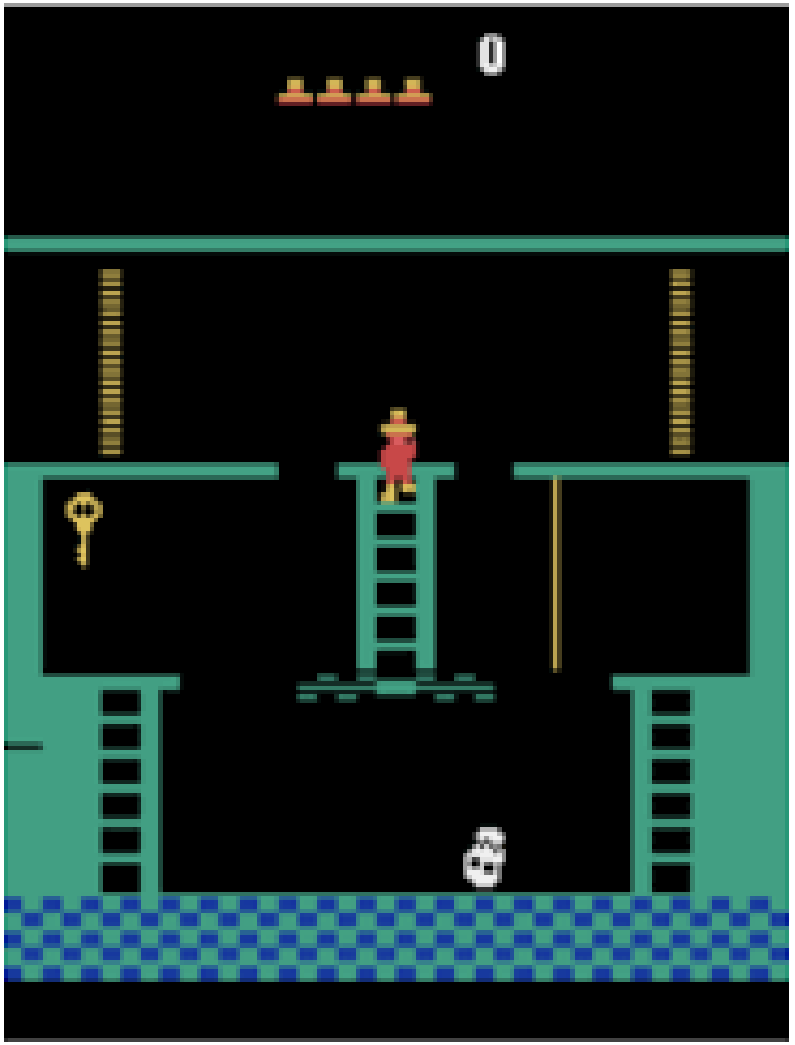
Algorithm 2 Batch actor-critic

for $i = 0, i < K, i++$ **do**
  sample $\{s_i, a_i\}_{i=1}^N \sim \pi_\theta(a|s)$ (run current policy)
  fit $\hat{V}_\phi^\pi$ to sampled reward sums
  evaluate $\hat{A}^\pi(s_i, a_i) = r(s_i, a_i) + \gamma \hat{V}_\phi^\pi(s_i') - \hat{V}_\phi^\pi(s_i)$
  $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i) \hat{A}^\pi(s_i, a_i)$
  $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Montezuma's Revenge game setting



Montezuma's Revenge game setting



Hierarchical reinforcement learning

Idea: have several levels of temporal abstractions. Most popular framework for 2 levels: *options framework*.

- *Meta-controller* a.k.a *Manager* learns policy over *sub-goals*
- *Controller* a.k.a *Worker* learns policy for achieving *sub-goals*

The main challenge: automatic discovery of sub-goals.

Hierachical approaches on Montezuma's Revenge

Hierachical DQN (h-DQN) [?]

- manually chosen objects (like door, ladder etc) are used as sub-goals
- controller is rewarded for reaching the object chosen by meta-controller
- meta-controller and controller have separate deep-Q networks
- at first stage meta-controller emits random sub-goals and only the controller is trained, then controller and meta-controller are trained jointly

Result: maximum average score of 375 is achieved after total 52.5 mln steps in the environment.

Pixel control [?]

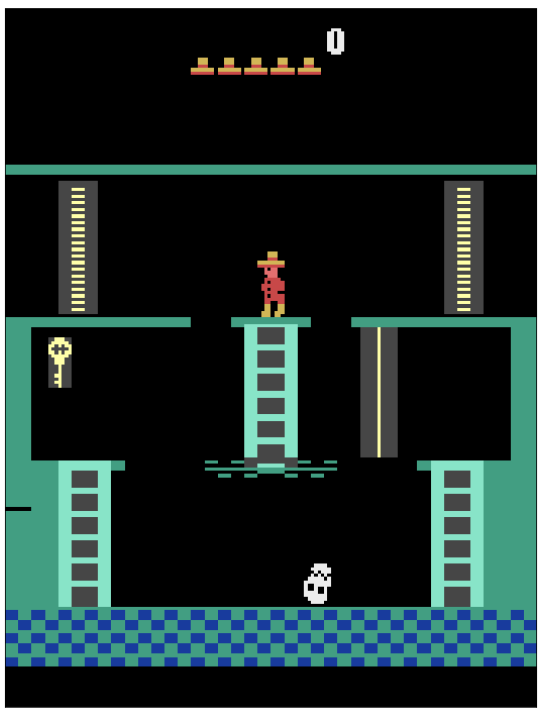
- image is divided by grid 6×6 into 36 patches. A sub-goal is movement in a chosen patch
- both controller and meta-controller trained by actor-critic
- controller and meta-control share perceptual module
- network architecture with LSTM module

Results: the best score of 400 is achieved after 20 million steps in the environment.

Hierarchical DQN (2016)

We have 2 neural networks: controller chooses native game actions, given observation and meta-controller's "order" and gets intrinsic reward for obeying meta-controller. Meta-controller chooses macro-actions given observation. Objects = actions for meta-controller:

- left bottom ladder
- right bottom ladder
- middle ladder
- rope
- key
- left door
- right door



Hierarchical actor-critic with LSTM architecture

1. At the beginning for a long time meta-actions chosen by the Manager are random, because the Manager learns only from the sparse extrinsic reward.
2. Random subgoals though allow the Worker to learn to navigate the room. Some objects like the middle ladder or the rope are easy to get to, therefore the reward signal that the Worker gets is not sparse. Since the Worker shares convolutional layers with the Manager, the Manager is also implicitly trained in this process.
3. At some point the Manager learns the right sequence of subgoals and the Worker successfully fulfills them. Everything works as it is supposed to.
4. However, following the outline of the pixel control experiment we give the Worker the shaped reward. The extrinsic reward remains larger than the intrinsic. Given the discount factor, it is more beneficial for the Worker to go directly to the key, even if the Manager (due to stochasticity needed for exploration) directs it to other objects.
5. But the Manager haven given a meaningless order, but still receiving the extrinsic reward increases state-action value. After a while he starts to give random orders.

Exploration in reinforcement learning

The full idea of reinforcement leaning – try it out, see if you like it, and if you do, try more of that in the future. *Ilya Sutskever, head of Tesla AI department*

Methods of local exploration in reinforcement learning:

- ϵ-greedy and Boltzmann exploration for DQN methods
- introducing *entropy* term in optimized functional for policy gradient methods
- noisy parameters exploration: sampling network weight from Gaussian distribution with learned parameters

Non-local methods – reward shaping, introducing *exploration bonus* term to reward for visiting states different from previously observed.

DQN with count-based exploration [?](NIPS)

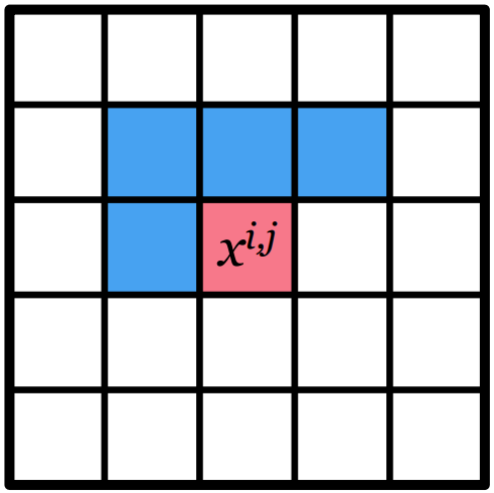


Figure 2: density model proposed – Context Tree Switching (CTS) density model based on a Bayesian variable-order Markov model. Takes as input an image and assigns to it a probability according to the product of location-dependent L-shaped filters, where the prediction of each filter is given by a CTS algorithm trained on past images.

Idea: derive *pseudo-counts* from density model on states $s \in S$
$\rho_n(x)$ – probability assigned by density model to state x given all the previous states
$\rho'_n(x)$ – an estimated probability of state x given x and all the previous states
$\hat{N}_n(x)$ – pseudo-count function of x
$\hat{n}$ – pseudo-count function total

$$\rho_n(x) = \frac{\hat{N}(x)}{\hat{n}}$$

$$\rho'_n(x) = \frac{\hat{N}_n(x)+1}{\hat{n}+1}$$

Exploration bonus:

$$r^{int} = 0.05\left(\hat{N}_n(x)+0.01\right)^{-1/2}$$

Result: the best average score of 3439 is achieved after 100 million frames.

Count-based exploration with neural density models [?](NIPS)

Lightened version of PixelCNN [?] used as the density model. Its core is a stack of 2 gated residual blocks with 16 feature maps (compared to 15 residual blocks with 128 feature maps in vanilla PixelCNN)

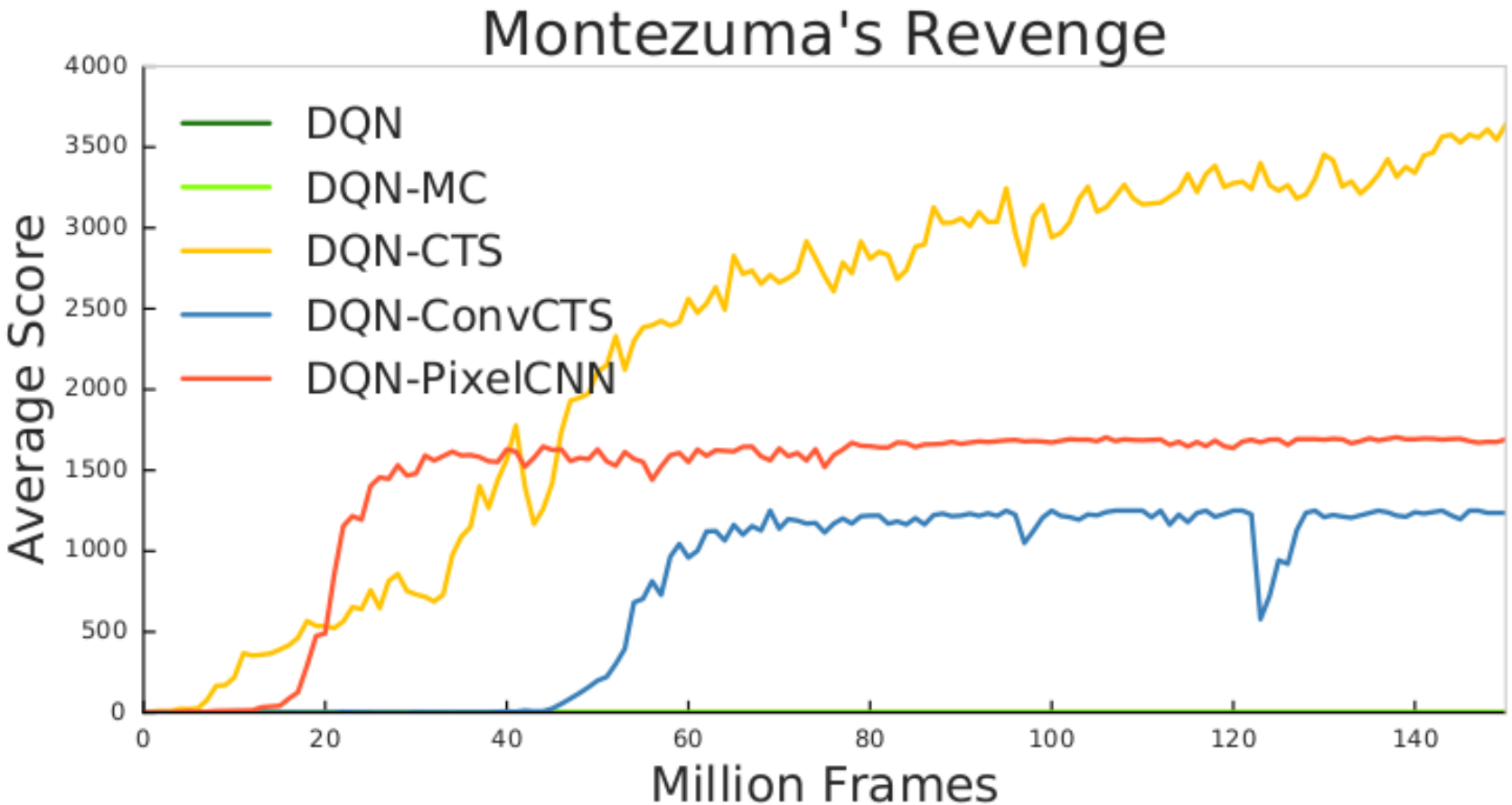


Figure 3: Comparison of performance of DQN agents trained with exploration bonus derived from different density models

Successes of RL

1. GO (initial version Alpha Go had DQN+bunch of hacks, Alpha Go is just DQN)
2. DOTA 2: *OpenAI Five plays 180 years worth of games against itself every day, learning via self-play. It trains using a scaled-up version of Proximal Policy Optimization running on 256 GPUs and 128,000 CPU cores — a larger-scale version of the system we built to play the much-simpler solo variant of the game last year. Using a separate LSTM for each hero and no human data, it learns recognizable strategies. This indicates that reinforcement learning can yield long-term planning with large but achievable scale — without fundamental advances, contrary to our own expectations upon starting the project.*
3. Some people minimise non-differentiable losses for NLP tasks(such as BLEU)
4. Neural Architecture Search: takes only 12800 examples which is extremely sample-efficient for RL