Deep Reinforcement Learning

Overview of main articles

Sergey Ivanov

August 25, 2018

MSU

Table of contents i

Reinforcement Learning [reminder]

MDP

MDP is $\{\mathbb{S}, \mathbb{A}, \mathbb{T}, r\}$:

$\mathbb{S}$ — set of states

$\mathbb{A}$ — set of actions

$\mathbb{T}$ — probability $p(s' \mid s, a)$, where $s, s' \in \mathbb{S}, a \in \mathbb{A}$

r — function $\mathbb{S} \to \mathbb{R}$

RL Goal

We search for policy $\pi : \mathbb{S} \to \mathbb{A}$ which maximizes[1]

$$\mathbb{E} \sum_t r(s_t)$$

[1] over what probability distributions is this expectation?

RL Goal

We search for policy $\pi : \mathbb{S} \to \mathbb{A}$ which maximizes[1]

$$\mathbb{E} \sum_t r(s_t)$$

This goal does not suit infinite horizon case, so for generalization purposes goal is substituted with

$$\mathbb{E} \sum_t \gamma^t r(s_t)$$

for $\gamma \in (0, 1)$.

[1] over what probability distributions is this expectation?

Definitions

For convenience[2]:

$$R = \sum_t \gamma^t r(s_t)$$

[2] *What does it depend on?*

Definitions

For convenience[2]:

$$R = \sum_t \gamma^t r(s_t)$$

For given policy π:

$$V^\pi(s) = \mathbb{E} R \mid s_0 = s$$

$$Q^\pi(s, a) = \mathbb{E} V(s') \mid s, a$$

[2] *What does it depend on?*

Definitions

For convenience[2]:

$$R = \sum_t \gamma^t r(s_t)$$

For given policy π:

$$V^\pi(s) = \mathbb{E} R \mid s_0 = s$$

$$Q^\pi(s, a) = \mathbb{E} V(s') \mid s, a$$

Let π^* be optimal policy.

[2] *What does it depend on?*

Bellman Equation

For every π it's true:

$$Q^{\pi}(s, a) = \mathbb{E}\left[r(s') + Q^{\pi}(s', \pi(s'))\right]$$

Bellman Equation

For every π it's true:

$$Q^{\pi}(s, a) = \mathbb{E}\left[r(s') + Q^{\pi}(s', \pi(s'))\right]$$

It's also true for π^*:

$$Q^{\pi^*}(s, a) = \mathbb{E}\left[r(s') + Q^{\pi^*}(s', \pi^*(s'))\right] \tag{1}$$

Bellman Equation

For every π it's true:

$$Q^{\pi}(s,a) = \mathbb{E}\left[r(s') + Q^{\pi}(s', \pi(s'))\right]$$

It's also true for π^*:

$$Q^{\pi^*}(s,a) = \mathbb{E}\left[r(s') + Q^{\pi^*}(s', \pi^*(s'))\right] \tag{1}$$

Note:

$$\pi^*(s) = \underset{a}{argmax}\, Q^{\pi^*}(s,a) \tag{2}$$

Bellman Equation

For every π it's true:

$$Q^{\pi}(s,a) = \mathbb{E}\left[r(s') + Q^{\pi}(s',\pi(s'))\right]$$

It's also true for π^*:

$$Q^{\pi^*}(s,a) = \mathbb{E}\left[r(s') + Q^{\pi^*}(s',\pi^*(s'))\right] \tag{1}$$

Note:

$$\pi^*(s) = \underset{a}{argmax}\, Q^{\pi^*}(s,a) \tag{2}$$

Bellman Equation

For every π it's true:

$$Q^{\pi}(s,a) = \mathbb{E}\left[r(s') + Q^{\pi}(s',\pi(s'))\right]$$

It's also true for π^*:

$$Q^{\pi^*}(s,a) = \mathbb{E}\left[r(s') + Q^{\pi^*}(s',\pi^*(s'))\right] \tag{1}$$

Note:

$$\pi^*(s) = \underset{a}{argmax}\, Q^{\pi^*}(s,a) \tag{2}$$

Insert (2) into (1):

Bellman Equation

$$Q^{\pi^*}(s,a) = \mathbb{E}\left[r(s') + Q^{\pi^*}(s', \underset{a}{argmax}\, Q^{\pi^*}(s',a))\right]$$

Bellman Equation

For every π it's true:

$$Q^{\pi}(s,a) = \mathbb{E}\left[r(s') + Q^{\pi}(s', \pi(s'))\right]$$

It's also true for π^*:

$$Q^{\pi^*}(s,a) = \mathbb{E}\left[r(s') + Q^{\pi^*}(s', \pi^*(s'))\right] \tag{1}$$

Note:

$$\pi^*(s) = \underset{a}{argmax}\, Q^{\pi^*}(s,a) \tag{2}$$

Insert (2) into (1):

Bellman Equation

$$Q^{\pi^*}(s,a) = \mathbb{E}\left[r(s') + \max_a Q^{\pi^*}(s', a)\right]$$

Temporal Difference Learning

For finite-state case Q^{π^*} is finite vector of unknown values.
Bellman equations can be solved using point iteration:

$$Q_{t+1}(s, a) = \mathbb{E}\left[r(s') + \max_a Q_t(s', a)\right]$$

Temporal Difference Learning

For finite-state case Q^{π^*} is finite vector of unknown values.
Bellman equations can be solved using point iteration:

$$Q_{t+1}(s, a) = \mathbb{E}\left[r(s') + \max_a Q_t(s', a)\right]$$

Problem: expectation.

Temporal Difference Learning

For finite-state case Q^{π^*} is finite vector of unknown values.
Bellman equations can be solved using point iteration:

$$Q_{t+1}(s, a) = \mathbb{E}\left[r(s') + \max_a Q_t(s', a)\right]$$

Problem: expectation.

Temporal Difference Learning

$$Q_{t+1}(s, a) = \alpha Q_t(s, a) + (1 - \alpha)\left[r(s') + \max_a Q_t(s', a)\right]$$

Temporal Difference Learning

For finite-state case Q^{π^*} is finite vector of unknown values.
Bellman equations can be solved using point iteration:

$$Q_{t+1}(s, a) = \mathbb{E}\left[r(s') + \max_a Q_t(s', a)\right]$$

Problem: expectation.

Temporal Difference Learning

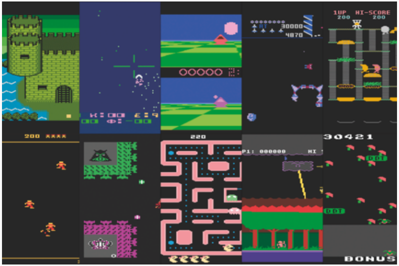
$$Q_{t+1}(s, a) = \alpha Q_t(s, a) + (1 - \alpha)\left[r(s') + \max_a Q_t(s', a)\right]$$

✓ Is a *contraction mapping* $\Rightarrow$ converges.

Deep Q-learning (2014)

Atari

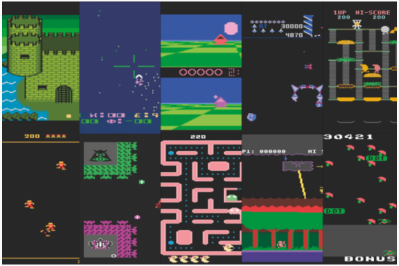
* No prepared features for each game.
* Screen image as input.
* Finite-state case... not quite finite.



Atari games

Atari

* No prepared features for each game.
* Screen image as input.
* Finite-state case... not quite finite.

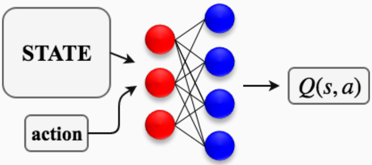


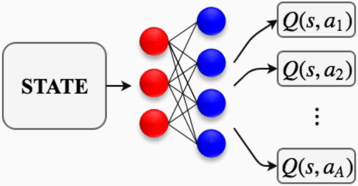
Atari games

We want to approximate $Q(s, a)$ with neural net.

Q-network

Option 1



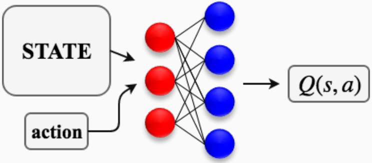


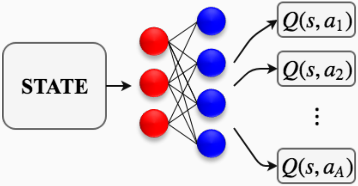
Option 2

Q-network

Option 1

Requires forward pass for each action[1]





Option 2

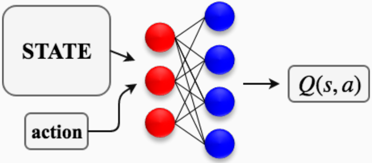
Number of actions must be adequate

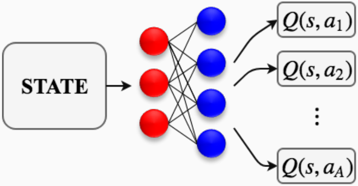
[1] Is there a case when option 1 might be better?

Q-network

Option 1

Requires forward pass for each action[1]





Option 2

Number of actions must be adequate

Atari: up to 18 discrete actions. Use option 2.

[1] Is there a case when option 1 might be better?

TD-learning to gradient descent

TD-learning is «similar» to gradient descent.

$$Q_{t+1}(s, a) = \alpha Q_t(s, a) + (1 - \alpha)\left[r(s') + \max_a Q_t(s', a)\right]$$

TD-learning to gradient descent

TD-learning is «similar» to gradient descent.

$$Q_{t+1}(s,a) = \alpha Q_t(s,a) + (1-\alpha)\left[r(s') + \max_a Q_t(s',a)\right] =$$

$$= Q_t(s,a) + (1-\alpha)\left[r(s') + \max_a Q_t(s',a) - Q_t(s,a)\right]$$

TD-learning to gradient descent

TD-learning is «similar» to gradient descent.

$$
\begin{aligned}
Q_{t+1}(s, a) = \alpha Q_t(s, a) + (1 - \alpha) \left[r(s') + \max_a Q_t(s', a) \right] = \\
= Q_t(s, a) + (1 - \alpha) \left[r(s') + \max_a Q_t(s', a) - Q_t(s, a) \right] = \\
= Q_t(s, a) - \eta \nabla_Q L
\end{aligned}
$$

TD-learning to gradient descent

TD-learning is «similar» to gradient descent.

$$
\begin{aligned}
Q_{t+1}(s,a) = \alpha Q_t(s,a) + (1-\alpha)\left[r(s') + \max_a Q_t(s',a)\right] = \\
= Q_t(s,a) + (1-\alpha)\left[r(s') + \max_a Q_t(s',a) - Q_t(s,a)\right] = \\
= Q_t(s,a) - \eta \nabla_Q L
\end{aligned}
$$

TD-learning to gradient descent

TD-learning is «similar» to gradient descent.

$$
\begin{aligned}
Q_{t+1}(s,a) = \alpha Q_t(s,a) + (1-\alpha)\left[r(s') + \max_a Q_t(s',a)\right] = \\
= Q_t(s,a) + (1-\alpha)\left[r(s') + \max_a Q_t(s',a) - Q_t(s,a)\right] = \\
= Q_t(s,a) - \eta \nabla_Q L
\end{aligned}
$$

Let $y = r(s') + \max_a Q_t(s',a)$.

TD-learning to gradient descent

TD-learning is «similar» to gradient descent.

$$\begin{aligned} Q_{t+1}(s,a) = \alpha Q_t(s,a) + (1-\alpha)\left[r(s') + \max_a Q_t(s',a)\right] = \\ = Q_t(s,a) + (1-\alpha)\left[r(s') + \max_a Q_t(s',a) - Q_t(s,a)\right] = \\ = Q_t(s,a) - \eta \nabla_Q L \end{aligned}$$

Let $y = r(s') + \max_a Q_t(s', a)$.
If dependence of y from Q is ignored:

$$L = (Q_t(s,a) - y)^2$$

Q-learning

With $Q(s, a)$ as neural net, its parameters θ determine function.

$$Q_{t+1}(s, a, \theta) = Q_t(s, a, \theta) - \eta \nabla_Q L$$

Q-learning

With $Q(s,a)$ as neural net, its parameters θ determine function.

$$Q_{t+1}(s,a,\theta) = Q_t(s,a,\theta) - \eta \nabla_Q L$$

Let's move gradient descent from space of Q functions to θ!

$$\theta_{t+1} = \theta_t - \beta \nabla_\theta L$$

Q-learning

With $Q(s, a)$ as neural net, its parameters θ determine function.

$$Q_{t+1}(s, a, \theta) = Q_t(s, a, \theta) - \eta \nabla_Q L$$

Let's move gradient descent from space of Q functions to θ!

$$\theta_{t+1} = \theta_t - \beta \nabla_\theta L$$

Problems:

× batch_size = 1. Wow.

Q-learning

With $Q(s, a)$ as neural net, its parameters θ determine function.

$$Q_{t+1}(s, a, \theta) = Q_t(s, a, \theta) - \eta \nabla_Q L$$

Let's move gradient descent from space of Q functions to θ!

$$\theta_{t+1} = \theta_t - \beta \nabla_\theta L$$

Problems:

- × batch_size = 1. Wow.
- × Target y changes after each step.

Q-learning

With $Q(s, a)$ as neural net, its parameters θ determine function.

$$Q_{t+1}(s, a, \theta) = Q_t(s, a, \theta) - \eta \nabla_Q L$$

Let's move gradient descent from space of Q functions to θ!

$$\theta_{t+1} = \theta_t - \beta \nabla_\theta L$$

Problems:

- $\times$ batch_size $= 1$. Wow.
- $\times$ Target y changes after each step.
- $\times$ All theoretical guarantees are lost.

ExperienceReplay

Utilize all experienced transitions $(s, a, s', r, done)$ for generating a batch for stochastic optimization step.

Utilize all experienced transitions $(s, a, s', r, done)$ for generating a batch for stochastic optimization step.

Pretend on each step that loss function is

$$\mathbb{E}_{(s,a,s',r,done)}(Q(s,a,\theta) - y(s',r,done))^2$$

Batch of transitions is sampled uniformly from memory.

ExperienceReplay

Utilize all experienced transitions $(s, a, s', r, done)$ for generating a batch for stochastic optimization step.

Pretend on each step that loss function is

$$\mathbb{E}_{(s,a,s',r,done)}(Q(s, a, \theta) - y(s', r, done))^2$$

Batch of transitions is sampled uniformly from memory.

✓ Decorellates samples.

* Target y can be calculated only for this batch.
* Only last N observed transitions may be stored

ε-greedy exploration

Problem: at the very beginning trajectories generated by $\pi(s) = \underset{a}{argmax}\, Q(s, a, \theta)$ are very similar.

ε-greedy exploration

Problem: at the very beginning trajectories generated by $\pi(s) = \underset{a}{argmax}\, Q(s, a, \theta)$ are very similar.

Choose random actions sometimes.

For example, with probability ε.

ε annealing

ε should be big at the beginning and small at the end.

ε annealing

ε should be big at the beginning and small at the end.

Atari: $\varepsilon(i) = 0.01 + 0.99\, exp\{-\frac{i}{30000}\}$ where i is frames counter.

Details

- Gray-scale frames were downsampled and cropped to 84x84.
- Last 4 frames[3] were considered as state to satisfy MDP Markov's property.
- Same NN architecture was used for all games: 3 convolutional[4] and 2 feedforward layers.

[3] 3 for Space Invaders cause of laser blinking period
[4] why no max pooling here?

More details

Playing Atari with Deep Reinforcement Learning (2014)

- Reward was restricted to $\{+1, 0, -1\}$. Allowed to use same learning rate for all games.

:(50 hours per game / 10 000 000 frames per game.

:} Bought by Google after 7 games.

Stabilizing Q-learning

Unstability

Recall our target on each step:

$$y(s', r) = r + \max_{a'} Q(s', a', \theta)$$

- Changes each frame
- Formally depends on θ
- "Correlates" with actions chosen during playing
- Tends to overestimate true $V(s')$

$\Rightarrow$ loss is completely unstable and can even diverge.

Target network (2015)

Change the target not every step, but each K-th step.

Target network (2015)

Change the target not every step, but each K-th step.

For this purpose:

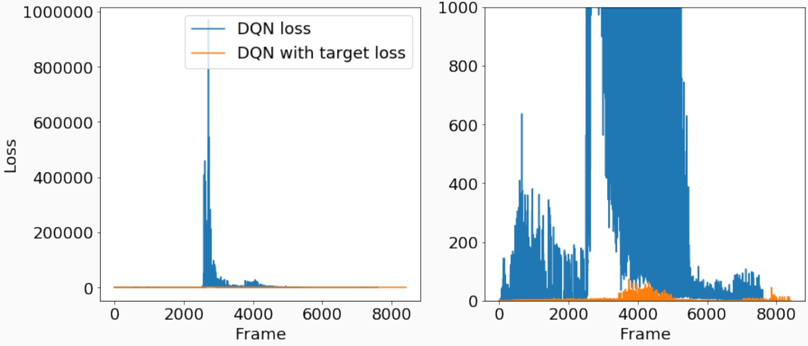
- Make a copy of Q-network, *target network*, with parameters θ^-
- Use it on every step to calculate

$$y(s', r) = r + \max_{a'} Q^{\text{target}}(s', a', \theta^-)$$

- Each K-th step update θ^- with current Q-network's weights θ.

Can be seen on loss

✓ Loss really stabilized!



Value overestimation

Recall our target is proxy of $V^{\pi^*}(s', a')$

$$y(s', r) = r + \max_{a'} Q(s', a', \theta)$$

Practice: this proxy overestimates true value of states.

Intuition: this max operator will prefer actions, for which $Q(s', a', \theta)$ is overestimating true value due to approximation or luck.

Action Selection vs Evaluation

Recall Bellman Equation derivation and untangle our target:

$$y(s', r) = r + \max_{a'} Q(s', a', \theta) = r + Q(s', argmax_{a'} Q(s', a', \theta), \theta)$$

Action Selection vs Evaluation

Recall Bellman Equation derivation and untangle our target:

$$y(s', r) = r + \max_{a'} Q(s', a', \theta) = r + Q(s', \underset{a'}{argmax}\, Q(s', a', \theta), \theta)$$

* $a' = \underset{a'}{argmax}\, Q(s', a', \theta)$ is *action selection*
* $Q(s', a', \theta)$ is *action evaluation*

Action Selection vs Evaluation

Recall Bellman Equation derivation and untangle our target:

$$y(s', r) = r + \max_{a'} Q(s', a', \theta) = r + Q(s', \underset{a'}{argmax}\, Q(s', a', \theta), \theta)$$

* $a' = \underset{a'}{argmax}\, Q(s', a', \theta)$ is *action selection*
* $Q(s', a', \theta)$ is *action evaluation*

General idea:

Use different approximations for evaluation and for selection to avoid *max*.

Two Q-learnings

Basic way to do this:
run two Q-learning algorithms with two approximations of Q^{π^*}:
$Q_1(s, a, \theta_1)$ and $Q_2(s, a, \theta_2)$.

Two Q-learnings

Basic way to do this:
run two Q-learning algorithms with two approximations of Q^{π^*}:
$Q_1(s, a, \theta_1)$ and $Q_2(s, a, \theta_2)$.

Targets for Q-learnings:

$$y_1 = r + Q_2(s', \underset{a'}{argmax}\, Q_1(s', a', \theta_1), \theta_2)$$

$$y_2 = r + Q_1(s', \underset{a'}{argmax}\, Q_2(s', a', \theta_2), \theta_1)$$

Double DQN (2015)

Deep Reinforcement Learning with Double Q-learning (2015)

- **more convenient way to do this:**

> Use target network as one of two approximations.

[5] how many backwards?

Double DQN (2015)

Deep Reinforcement Learning with Double Q-learning (2015)

- **more convenient way to do this:**

Use target network as one of two approximations.

$$y = r + Q^{\text{target}}(s', \underset{a'}{argmax}\, Q(s', a', \theta), \theta^{-})$$

[5] how many backwards?

Double DQN (2015)

Deep Reinforcement Learning with Double Q-learning (2015)
- **more convenient way to do this:**

Use target network as one of two approximations.

$$y = r + Q^{\text{target}}(s', \underset{a'}{argmax}\, Q(s', a', \theta), \theta^{-})$$

* Keep ignoring dependence of y from θ.

* Requires three forward passes on each step[5].

[5] how many backwards?

Comparing DQNs

Table 1: DQN targets

DQN	target y
Classic Deep Q-learning	$r + Q(s', \underset{a'}{argmax}\, Q(s', a', \theta), \theta)$
With target-network	$r + Q^{\text{target}}(s', \underset{a'}{argmax}\, Q^{\text{target}}(s', a', \theta^{-}), \theta^{-})$
Double Deep Q-learning	$r + Q^{\text{target}}(s', \underset{a'}{argmax}\, Q(s', a', \theta), \theta^{-})$

Dueling DQN: Motivation

Note:

* In most states our choice of action does not affect future value.
* After finding $Q(s, a)$ Q-learning still gains no information about $Q(s, a')$ for $a' \neq a$.

Dueling DQN: Motivation

Note:

* In most states our choice of action does not affect future value.
* After finding $Q(s, a)$ Q-learning still gains no information about $Q(s, a')$ for $a' \neq a$.

$\Rightarrow$ after trying an action in a bad state, Q-learning wants to try all other actions in this state.

Dueling DQN: Motivation

Note:

* In most states our choice of action does not affect future value.
* After finding $Q(s, a)$ Q-learning still gains no information about $Q(s, a')$ for $a' \neq a$.

$\Rightarrow$ after trying an action in a bad state, Q-learning wants to try all other actions in this state.

> Learning $Q(s, a)$ should lead to learning $V(s)$

Advantage function

Define *advantage* function:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$

Advantage function

Define *advantage* function:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

Note:

$$\begin{aligned}\mathbb{E}_{a\sim\pi} A^\pi(s,a) &= \mathbb{E}_{a\sim\pi} Q^\pi(s,a) - V^\pi(s) = \\ &= \mathbb{E}_{a\sim\pi} Q^\pi(s,a) - \mathbb{E}_{a\sim\pi} Q^\pi(s,a) = 0\end{aligned}$$

Advantage function

Define *advantage* function:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$

Note:

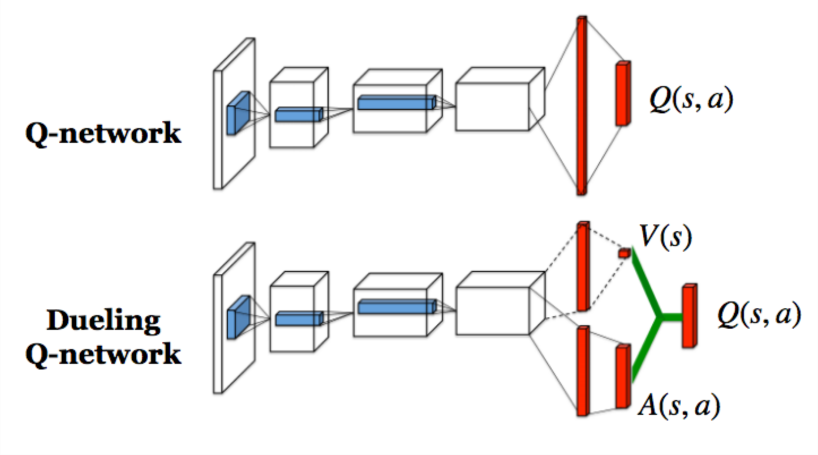
$$\begin{aligned}\mathbb{E}_{a\sim\pi} A^\pi(s, a) &= \mathbb{E}_{a\sim\pi} Q^\pi(s, a) - V^\pi(s) = \\ &= \mathbb{E}_{a\sim\pi} Q^\pi(s, a) - \mathbb{E}_{a\sim\pi} Q^\pi(s, a) = 0\end{aligned}$$

Rewrite Q-function in terms of value of state:

$$Q^\pi(s, a) = V^\pi(s) + A^\pi(s, a)$$

Dueling DQN (2016)

Dueling Network Architectures for Deep Reinforcement Learning (2016)



Dueling Q-network architecture

Struggling with identifiability

Problem: $A(s, a)$ is not arbitrary. Recall $\mathbb{E}_{a \sim \pi} A^\pi(s, a) = 0$.

Struggling with identifiability

Problem: $A(s, a)$ is not arbitrary. Recall $\mathbb{E}_{a \sim \pi} A^{\pi}(s, a) = 0$.

For deterministic $\pi(s) = \underset{a}{argmax}\, Q(s, a)$ this is equivalent to

$$\max_a A(s, a) = 0$$

Struggling with identifiability

Problem: $A(s,a)$ is not arbitrary. Recall $\mathbb{E}_{a\sim\pi} A^{\pi}(s,a) = 0$.

For deterministic $\pi(s) = \underset{a}{\mathit{argmax}}\, Q(s,a)$ this is equivalent to

$$\max_a A(s,a) = 0$$

Proposition:

$$Q(s,a) = V(s) + A(s,a) - \max_a A(s,a)$$

Struggling with identifiability

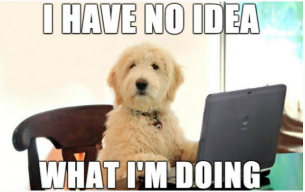
Problem: $A(s,a)$ is not arbitrary. Recall $\mathbb{E}_{a\sim\pi} A^\pi(s,a) = 0$.

For deterministic $\pi(s) = \underset{a}{argmax}\, Q(s,a)$ this is equivalent to

$$\max_a A(s,a) = 0$$

Proposition:

$$Q(s,a) = V(s) + A(s,a) - \max_a A(s,a)$$



$$Q(s,a) = V(s) + A(s,a) - \underset{a}{mean}\, A(s,a)$$

suddenly worked better.

Results

✓ Learning $Q(s, a)$ leads to correcting $V(s)$.

* Only network architecture is changed.

* Double DQN still works for dueling architecture.

Prioritized replay memory (2015)

Motivation

In standard DQN we sample batch of transitions from replay memory uniformly.

- × Some transitions are more important than others
- × Replay memory is full of almost useless transitions

Motivation

In standard DQN we sample batch of transitions from replay memory uniformly.

- × Some transitions are more important than others
- × Replay memory is full of almost useless transitions

$\delta = |y(s', r, done) - Q(s, a)|$ is a good proxy of transition importance

Prioritized Replay Memory (2015)

Prioritized Experience Replay (2015):

$$p(\mathcal{T}) \propto \delta(\mathcal{T})^{\alpha}$$

Authors found $\alpha \approx 0.6$ is a good universal value.

Prioritized Replay Memory (2015)

Prioritized Experience Replay (2015):

$$p(\mathcal{T}) \propto \delta(\mathcal{T})^{\alpha}$$

Authors found $\alpha \approx 0.6$ is a good universal value.

Problems:

- × On each step this probability changes for all the replay memory [6]

[6] which capacity is on the order of 1M transitions

Prioritized Replay Memory (2015)

Prioritized Experience Replay (2015):

$$p(\mathcal{T}) \propto \delta(\mathcal{T})^{\alpha}$$

Authors found $\alpha \approx 0.6$ is a good universal value.

Problems:

- $\times$ On each step this probability changes for all the replay memory [6]
  - $\approx$ on each step update δ only for the sampled batch used for learning

[6] which capacity is on the order of 1M transitions

Prioritized Replay Memory (2015)

Prioritized Experience Replay (2015):

$$p(\mathcal{T}) \propto \delta(\mathcal{T})^{\alpha}$$

Authors found $\alpha \approx 0.6$ is a good universal value.

Problems:

- $\times$ On each step this probability changes for all the replay memory [6]
  - $\approx$ on each step update δ only for the sampled batch used for learning
- $\times$ Introduces bias (transitions are now sampled from hell knows what distribution).

[6] which capacity is on the order of 1M transitions

Background: Importance Sampling

For arbitrary distribution $q(x)$:

$$\begin{aligned}\mathbb{E}_{p(x)} f(x) = \int p(x) f(x) dx = \int \frac{q(x)}{q(x)} p(x) f(x) dx = \\ = \int q(x) \frac{p(x)}{q(x)} f(x) dx = \mathbb{E}_{q(x)} \frac{p(x)}{q(x)} f(x)\end{aligned}$$

Background: Importance Sampling

For arbitrary distribution $q(x)$:

$$\mathbb{E}_{p(x)} f(x) = \int p(x) f(x) dx = \int \frac{q(x)}{q(x)} p(x) f(x) dx =$$
$$= \int q(x) \frac{p(x)}{q(x)} f(x) dx = \mathbb{E}_{q(x)} \frac{p(x)}{q(x)} f(x)$$

That's exactly what we want: to substitute expectation of loss over uniform sampling from experience replay to expectation over our own prioritized distribution!

Background: Importance Sampling

For arbitrary distribution $q(x)$:

$$\mathbb{E}_{p(x)} f(x) = \int p(x) f(x) dx = \int \frac{q(x)}{q(x)} p(x) f(x) dx =$$
$$= \int q(x) \frac{p(x)}{q(x)} f(x) dx = \mathbb{E}_{q(x)} \frac{p(x)}{q(x)} f(x)$$

That's exactly what we want: to substitute expectation of loss ($f(x)$) over uniform sampling from experience replay ($p(x)$) to expectation over our own prioritized distribution ($q(x)$) !

Applying Importance Sampling

If N is replay memory capacity:

$$L = \mathbb{E}_{\mathcal{T} \sim uniform}(y - Q(s,a))^2 = \mathbb{E}_{\mathcal{T} \sim prioritized} \frac{1}{Np(\mathcal{T})}(y - Q(s,a))^2$$

IS just adds weights to our batch:

$$w_i = \frac{1}{Np(\mathcal{T}_i)}$$

Annealing weights

Problem: at the beginning these weights might not be that relevant, yet slowing down learning.

Annealing weights

Problem: at the beginning these weights might not be that relevant, yet slowing down learning.

Let's smooth them at the beginning of learning:

$$L = \mathbb{E}_{\mathcal{T} \sim prioritized} \left(\frac{1}{Np(\mathcal{T})} \right)^{\beta} (y - Q(s, a))^2,$$

where β changes from 0.4 to 1 linearly during first 100 000 frames.

Hints

* Weights significantly vary scale of loss function. Constant learning rate might be inappropriate.

  Hint:[7] normalize weights by dividing on $\max_i w_i$.

[7] max taken over all replay memory. Yet in some implementations it is taken over current batch

Hints

* Weights significantly vary scale of loss function. Constant learning rate might be inappropriate.

  Hint:[7] normalize weights by dividing on $\max_i w_i$.

* $\min(1, |\delta|)$ is used instead of $|\delta|$ for stabilization purposes.

[7] max taken over all replay memory. Yet in some implementations it is taken over current batch

Hints

* Weights significantly vary scale of loss function. Constant learning rate might be inappropriate.

    Hint:[7] normalize weights by dividing on $\max_i w_i$.

* $\min(1, |\delta|)$ is used instead of $|\delta|$ for stabilization purposes.
* new transitions are stored with maximum priority.

[7] max taken over all replay memory. Yet in some implementations it is taken over current batch

Noisy networks for exploration (2017)

Noisy Nets (2017)

Problem: ε-greedy exploration is *state-independent*.

Noisy Nets (2017)

Problem: ε-greedy exploration is *state-independent*.

Add parametric noise to the weights of Q-network

Noisy Nets (2017)

Problem: ε-greedy exploration is *state-independent*.

Add parametric noise to the weights of Q-network

Noisy Nets for Exploration (2017):

$$w_i = \mu_i + \sigma_i * \varepsilon_i, \quad \varepsilon \sim \mathcal{N}(0, 1)$$

* μ_i, σ_i are both learnable parameters.
* all weights are independent random variables
* use policy $\pi(s) = \underset{a}{argmax}\, Q(s, a, \mu, \sigma, \varepsilon)$

Optimized Loss

Formally, our loss[8] is now:

$$\mathbb{E}_\varepsilon \mathbb{E}_\mathcal{T}(Q(s, a, \theta, \varepsilon) - y(\mathcal{T}))^2$$

[8]Noisy Net is not a bayesian NN as it does not model probability; loss minimization is also not an upper bound optimization

Optimized Loss

Formally, our loss[8] is now:

$$\mathbb{E}_{\varepsilon}\mathbb{E}_{\mathcal{T}}(Q(s, a, \theta, \varepsilon) - y(\mathcal{T}))^2$$

Problem: y also depends on stochastic Q-function.

[8]Noisy Net is not a bayesian NN as it does not model probability; loss minimization is also not an upper bound optimization

Optimized Loss

Formally, our loss[8] is now:

$$\mathbb{E}_{\varepsilon}\mathbb{E}_{\mathcal{T}}(Q(s, a, \theta, \varepsilon) - y(\mathcal{T}))^2$$

Problem: y also depends on stochastic Q-function.

* use different noise samples for it:

$$y = r + Q(s', \underset{a'}{argmax}\, Q(s', a', \varepsilon''), \varepsilon')$$

[8]Noisy Net is not a bayesian NN as it does not model probability; loss minimization is also not an upper bound optimization

Noise generation optimization

Problem: noise generation turns to be a bottleneck in terms of wall-clock time.

- $MN + N$ samples required for linear layer mapping M features to N.

Noise generation optimization

Problem: noise generation turns to be a bottleneck in terms of wall-clock time.

- $MN + N$ samples required for linear layer mapping M features to N.

Alternative:

- generate M noises ε_m and N noises ε_n
- consider weight noise $\varepsilon_{mn} = f(\varepsilon_m)f(\varepsilon_n)$, where f is scaling function (signed square root)

Noise generation optimization

Problem: noise generation turns to be a bottleneck in terms of wall-clock time.

- $MN + N$ samples required for linear layer mapping M features to N.

Alternative:

- generate M noises ε_m and N noises ε_n
- consider weight noise $\varepsilon_{mn} = f(\varepsilon_m)f(\varepsilon_n)$, where f is scaling function (signed square root)
- N more noises for bias [9]

[9] authors also scale them with f

Noise generation optimization

Problem: noise generation turns to be a bottleneck in terms of wall-clock time.

- $MN + N$ samples required for linear layer mapping M features to N.

Alternative:

- generate M noises ε_m and N noises ε_n
- consider weight noise $\varepsilon_{mn} = f(\varepsilon_m)f(\varepsilon_n)$, where f is scaling function (signed square root)
- N more noises for bias [9]

✓ just $M + 2N$ noise samples are needed.

[9] authors also scale them with f

Noise generation optimization

Problem: noise generation turns to be a bottleneck in terms of wall-clock time.

- $MN + N$ samples required for linear layer mapping M features to N.

Alternative:

- generate M noises ε_m and N noises ε_n
- consider weight noise $\varepsilon_{mn} = f(\varepsilon_m)f(\varepsilon_n)$, where f is scaling function (signed square root)
- N more noises for bias [9]

✓ just $M + 2N$ noise samples are needed.

  * for whole batch![10]

---

[9] authors also scale them with f

[10] is this theoretically coherent?

Application tips

✓ No hyperparameters

  * Except where to put noise in the network... Convolution layers better leave deterministic[11].

[11] why?

Application tips

✓ No hyperparameters

  * Except where to put noise in the network... Convolution layers better leave deterministic[11].

✓ noise magnitude σ will (hopefully[12]) vanish state-dependently through the learning process

[11] why?

[12] on practice, behaves very differently from game to game

Application tips

- ✓ No hyperparameters
  - * Except where to put noise in the network... Convolution layers better leave deterministic[11].
- ✓ noise magnitude σ will (hopefully[12]) vanish state-dependently through the learning process
  - * yet $w_i = \mu_i$ can be used for exploitation purposes

[11] why?
[12] on practice, behaves very differently from game to game

Application tips

- ✓ No hyperparameters
  - * Except where to put noise in the network... Convolution layers better leave deterministic[11].
- ✓ noise magnitude σ will (hopefully[12]) vanish state-dependently through the learning process
  - * yet $w_i = \mu_i$ can be used for exploitation purposes
- ✓ almost random behavior at the beginning

[11] why?

[12] on practice, behaves very differently from game to game

Application tips

- ✓ No hyperparameters
  - * Except where to put noise in the network... Convolution layers better leave deterministic[11].
- ✓ noise magnitude σ will (hopefully[12]) vanish state-dependently through the learning process
  - * yet $w_i = \mu_i$ can be used for exploitation purposes
- ✓ almost random behavior at the beginning
  - * yet ε-greedy strategy may also be used

[11] why?

[12] on practice, behaves very differently from game to game

Categorical DQN (2017)

Motivation

Consider a state where you get 1000 or -1000 with probabilities 0.5.

Motivation

Consider a state where you get 1000 or -1000 with probabilities 0.5.
Q-network would say value of state is 0.

Motivation

Consider a state where you get 1000 or -1000 with probabilities 0.5.
Q-network would say value of state is 0. **But you never really get 0**.

Motivation

Consider a state where you get 1000 or -1000 with probabilities 0.5.
Q-network would say value of state is 0. **But you never really get 0**.

Learn a distribution over future reward instead of it's expectation.

Value Distribution

Recall

$$Q^\pi(s, a) = \mathbb{E} \sum_t r(s_t) \mid s, a$$

Value Distribution

Recall

$$Q^\pi(s, a) = \mathbb{E} \sum_t r(s_t) \mid s, a$$

A Distributional Perspective on Reinforcement Learning (2017):

For fixed policy π let's define *value distribution*:

Value distribution

Let's define value distribution as distribution of

$$Z^\pi(s, a) = \sum_t r(s_t) \mid s, a$$

Value Distribution

Recall

$$Q^\pi(s, a) = \mathbb{E} \sum_t r(s_t) \mid s, a$$

A Distributional Perspective on Reinforcement Learning (2017):

For fixed policy π let's define *value distribution*:

Value distribution

Let's define value distribution as distribution of

$$Z^\pi(s, a) = \sum_t r(s_t) \mid s, a$$

! It's a random variable!

Dynamic programming for value distribution

Value distribution satisfies a *recursive distributional equation*:

$$Z^\pi(s,a) \stackrel{\mathrm{D}}{=} r(s,a) + \gamma Z^\pi(s', \pi(s'))$$

Dynamic programming for value distribution

Value distribution satisfies a *recursive distributional equation*:

$$Z^\pi(s,a) \stackrel{\mathrm{D}}{=} r(s,a) + \gamma Z^\pi(s', \pi(s'))$$

Just equivalence of c.d.f. of left and right part :}

Dynamic programming for value distribution

Value distribution satisfies a *recursive distributional equation*:

$$Z^{\pi}(s,a) \overset{\mathrm{D}}{=} r(s,a) + \gamma Z^{\pi}(s', \pi(s'))$$

Just equivalence of c.d.f. of left and right part :}

Question: will point iteration be a contraction mapping for some metric in the space of value distributions?

Dynamic programming for value distribution

Value distribution satisfies a *recursive distributional equation*:

$$Z^{\pi}(s,a) \overset{\mathrm{D}}{=} r(s,a) + \gamma Z^{\pi}(s', \pi(s'))$$

Just equivalence of c.d.f. of left and right part :}

Question: will point iteration be a contraction mapping for some metric in the space of value distributions?

Dynamic programming for value distribution

Value distribution satisfies a *recursive distributional equation*:

$$Z^\pi(s,a) \overset{\mathrm{D}}{=} r(s,a) + \gamma Z^\pi(s', \pi(s'))$$

Just equivalence of c.d.f. of left and right part :}

Question: will point iteration be a contraction mapping for some metric in the space of value distributions?

✓ yes, for $d(Z_1, Z_2) = \sup_{s,a} \mathcal{W}(Z_1(s,a), Z_2(s,a))$, where $\mathcal{W}$ is Wasserstein distance between two random variables.

Distributional Bellman Equation

Analogically: $\pi^*(s) = \max_a \mathbb{E} Z^{\pi^*}(s, a)$

Distributional Bellman Equation

Analogically: $\pi^*(s) = \max_a \mathbb{E} Z^{\pi^*}(s, a)$

Distributional Bellman equation

$$Z^{\pi^*}(s, a) \overset{\mathrm{D}}{=} r(s, a) + \gamma Z^{\pi^*}(s', \pi^*(s'))$$

Distributional Bellman Equation

Analogically: $\pi^*(s) = \max_a \mathbb{E} Z^{\pi^*}(s, a)$

Distributional Bellman equation

$$Z^{\pi^*}(s, a) \stackrel{\mathrm{D}}{=} r(s, a) + \gamma Z^{\pi^*}(s', \max_{a'} \mathbb{E} Z^{\pi^*}(s', a'))$$

Distributional Bellman Equation

Analogically: $\pi^*(s) = \max_a \mathbb{E} Z^{\pi^*}(s, a)$

Distributional Bellman equation

$$Z^{\pi^*}(s, a) \overset{\mathrm{D}}{=} r(s, a) + \gamma Z^{\pi^*}(s', \max_{a'} \mathbb{E} Z^{\pi^*}(s', a'))$$

Question: will point iteration be a contraction mapping for some metric in the space of value distributions? [13]

[13] and why are we asking this again?

Distributional Bellman Equation

Analogically: $\pi^*(s) = \max_a \mathbb{E} Z^{\pi^*}(s, a)$

Distributional Bellman equation

$$Z^{\pi^*}(s, a) \overset{\mathrm{D}}{=} r(s, a) + \gamma Z^{\pi^*}(s', \max_{a'} \mathbb{E} Z^{\pi^*}(s', a'))$$

Question: will point iteration be a contraction mapping for some metric in the space of value distributions? [13]

$\times$ no, it will not.

[13] and why are we asking this again?

Distributional Bellman Equation

Analogically: $\pi^*(s) = \max_a \mathbb{E} Z^{\pi^*}(s, a)$

Distributional Bellman equation

$$Z^{\pi^*}(s, a) \overset{\mathrm{D}}{=} r(s, a) + \gamma Z^{\pi^*}(s', \max_{a'} \mathbb{E} Z^{\pi^*}(s', a'))$$

Question: will point iteration be a contraction mapping for some metric in the space of value distributions? [13]

- × no, it will not.
- × there may be no fixed point at all

[13] and why are we asking this again?

Distributional Bellman Equation

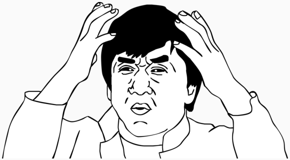
Analogically: $\pi^*(s) = \max_a \mathbb{E} Z^{\pi^*}(s, a)$

Distributional Bellman equation

$$Z^{\pi^*}(s, a) \stackrel{D}{=} r(s, a) + \gamma Z^{\pi^*}(s', \max_{a'} \mathbb{E} Z^{\pi^*}(s', a'))$$

Question: will point iteration be a contraction mapping for some metric in the space of value distributions? [13]

- $\times$ no, it will not.
- $\times$ there may be no fixed point at all
- $\times$ and existence of one doesn't guarantee convergence to it



[13] and why are we asking this again?

We'll do it anyway!

Let's do point iteration anyway! Our wish:

$$p(Z_{t+1}(s,a)) \leftarrow p\left(r(s,a) + \gamma Z_t\left[s', \max_{a'} \mathbb{E} Z_t(s',a')\right]\right)$$

We'll do it anyway!

Let's do point iteration anyway! Our wish:

$$p(Z_{t+1}(s,a)) \leftarrow p\left(r(s,a) + \gamma Z_t\left[s', \max_{a'} \mathbb{E} Z_t(s',a')\right]\right)$$

Problems:

- **?** $p(Z_t)$ is some distribution on $\mathbb{R}$. How do we represent it?

We'll do it anyway!

Let's do point iteration anyway! Our wish:

$$p(Z_{t+1}(s,a)) \leftarrow p\left(r(s,a) + \gamma Z_t\left[s', \max_{a'} \mathbb{E} Z_t(s',a')\right]\right)$$

Problems:

- **?** $p(Z_t)$ is some distribution on $\mathbb{R}$. How do we represent it?
  - ✓ use some distribution family as approximation: $p(Z_t(s,a)) \approx Z_\theta$

We'll do it anyway!

Let's do point iteration anyway! Our wish:

$$p(Z_{t+1}(s,a)) \leftarrow p\left(r(s,a) + \gamma Z_t\left[s', \max_{a'} \mathbb{E} Z_t(s',a')\right]\right)$$

Problems:

- **?** $p(Z_t)$ is some distribution on $\mathbb{R}$. How do we represent it?
  - ✓ use some distribution family as approximation: $p(Z_t(s,a)) \approx Z_\theta$
- **?** we have only samples of s' and $r(s,a)$

We'll do it anyway!

Let's do point iteration anyway! Our wish:

$$p(Z_{t+1}(s,a)) \leftarrow p\left(r(s,a) + \gamma Z_t\left[s', \max_{a'} \mathbb{E} Z_t(s',a')\right]\right)$$

Problems:

? $p(Z_t)$ is some distribution on $\mathbb{R}$. How do we represent it?

  ✓ use some distribution family as approximation: $p(Z_t(s,a)) \approx Z_\theta$

? we have only samples of s' and $r(s,a)$

  * in DQN case we optimized $L = \mathbb{E}_{\mathcal{T}}(y - Q_\theta)^2$

We'll do it anyway!

Let's do point iteration anyway! Our wish:

$$p(Z_{t+1}(s,a)) \leftarrow p\left(r(s,a) + \gamma Z_t\left[s', \max_{a'} \mathbb{E} Z_t(s',a')\right]\right)$$

Problems:

- **?** $p(Z_t)$ is some distribution on $\mathbb{R}$. How do we represent it?
  - ✓ use some distribution family as approximation: $p(Z_t(s,a)) \approx Z_\theta$
- **?** we have only samples of s' and $r(s,a)$
  - * in DQN case we optimized $L = \mathbb{E}_{\mathcal{T}}(y - Q_\theta)^2$
  - ✓ let's optimize $L = \mathcal{D}(p(Z_{t+1}) \parallel Z_\theta)$, where $\mathcal{D}$ is some divergence

We'll do it anyway!

Let's do point iteration anyway! Our wish:

$$p(Z_{t+1}(s,a)) \leftarrow p\left(r(s,a) + \gamma Z_t\left[s', \max_{a'} \mathbb{E} Z_t(s',a')\right]\right)$$

Problems:

- **?** $p(Z_t)$ is some distribution on $\mathbb{R}$. How do we represent it?
  - ✓ use some distribution family as approximation: $p(Z_t(s,a)) \approx Z_\theta$
- **?** we have only samples of s' and $r(s,a)$
  - * in DQN case we optimized $L = \mathbb{E}_{\mathcal{T}}(y - Q_\theta)^2$
  - ✓ let's optimize $L = \mathcal{D}(p(Z_{t+1}) \parallel Z_\theta)$, where $\mathcal{D}$ is some divergence

Loss

$Z_{t+1} \overset{\mathrm{D}}{=} r(s,a) + \gamma Z_t(s', \max_a \mathbb{E} Z_t(s,a))$ is a convolution involving MDP transition probability $p(s' \mid s,a)$:

$$p(Z_{t+1}) = \sum_{s'} p(y \mid s') p(s' \mid s,a)$$

where $y(s') = r(s') + \gamma Z_t(s', \max_a \mathbb{E} Z_t(s,a))$.

Loss

$Z_{t+1} \stackrel{\mathrm{D}}{=} r(s,a) + \gamma Z_t(s', \max_a \mathbb{E} Z_t(s,a))$ is a convolution involving MDP transition probability $p(s' \mid s,a)$:

$$p(Z_{t+1}) = \sum_{s'} p(y \mid s') p(s' \mid s,a)$$

where $y(s') = r(s') + \gamma Z_t(s', \max_a \mathbb{E} Z_t(s,a))$.

Note: for fixed $Z_t(s,a)$ and given s' function $y(s')$ is deterministic!

Loss

$Z_{t+1} \stackrel{\mathrm{D}}{=} r(s,a) + \gamma Z_t(s', \max_a \mathbb{E} Z_t(s,a))$ is a convolution involving MDP transition probability $p(s' \mid s, a)$:

$$p(Z_{t+1}) = \sum_{s'} p(y \mid s') p(s' \mid s, a)$$

where $y(s') = r(s') + \gamma Z_t(s', \max_a \mathbb{E} Z_t(s,a))$.

Note: for fixed $Z_t(s,a)$ and given s' function $y(s')$ is deterministic!

$\Rightarrow$ for given $p(Z_t(s,a))$ we can get $p(y \mid s')$

Loss

$Z_{t+1} \stackrel{\mathrm{D}}{=} r(s,a) + \gamma Z_t(s', \max_a \mathbb{E} Z_t(s,a))$ is a convolution involving MDP transition probability $p(s' \mid s,a)$:

$$p(Z_{t+1}) = \sum_{s'} p(y \mid s') p(s' \mid s,a)$$

where $y(s') = r(s') + \gamma Z_t(s', \max_a \mathbb{E} Z_t(s,a))$.

Note: for fixed $Z_t(s,a)$ and given s' function $y(s')$ is deterministic!

$\Rightarrow$ for given $p(Z_t(s,a))$ we can get $p(y \mid s')$

Problem: and what?

Loss

$Z_{t+1} \overset{\mathrm{D}}{=} r(s,a) + \gamma Z_t(s', \max_a \mathbb{E} Z_t(s,a))$ is a convolution involving MDP transition probability $p(s' \mid s, a)$:

$$p(Z_{t+1}) = \sum_{s'} p(y \mid s') p(s' \mid s, a) = \mathbb{E}_{\mathcal{T}} p(y \mid s')$$

where $y(s') = r(s') + \gamma Z_t(s', \max_a \mathbb{E} Z_t(s,a))$.

Note: for fixed $Z_t(s,a)$ and given s' function $y(s')$ is deterministic!

$\Rightarrow$ for given $p(Z_t(s,a))$ we can get $p(y \mid s')$

Problem: and what?

KL-magic

Our loss is $L = \mathcal{D}(p(Z_{t+1}) \parallel Z_\theta) = \mathcal{D}(\mathbb{E}_{\mathcal{T}}\, p(y \mid s') \parallel Z_\theta)$.

KL-**magic**

Our loss is $L = \mathcal{D}(p(Z_{t+1}) \parallel Z_\theta) = \mathcal{D}(\mathbb{E}_{\mathcal{T}} p(y \mid s') \parallel Z_\theta)$.

* we are free to choose $\mathcal{D}$!

KL-magic

Our loss is $L = \mathcal{D}(p(Z_{t+1}) \parallel Z_\theta) = \mathcal{D}(\mathbb{E}_\mathcal{T} p(y \mid s') \parallel Z_\theta)$.

* we are free to choose $\mathcal{D}$!

$$\mathrm{KL}(p(Z_{t+1}) \parallel Z_\theta) = -\int p(Z_{t+1}) \log Z_\theta + C = -\int \mathbb{E}_\mathcal{T} p(y \mid s') \log Z_\theta + C =$$

KL-**magic**

Our loss is $L = \mathcal{D}(p(Z_{t+1}) \parallel Z_\theta) = \mathcal{D}(\mathbb{E}_\mathcal{T} p(y \mid s') \parallel Z_\theta)$.

* we are free to choose $\mathcal{D}$!

$$\mathrm{KL}(p(Z_{t+1}) \parallel Z_\theta) = -\int p(Z_{t+1}) \log Z_\theta + C = -\int \mathbb{E}_\mathcal{T} p(y \mid s') \log Z_\theta + C =$$

$$= -\mathbb{E}_\mathcal{T} \int p(y \mid s') \log Z_\theta + C = \mathbb{E}_\mathcal{T} \, \mathrm{KL}(p(y \mid s') \parallel Z_\theta) + C$$

KL-magic

Our loss is $L = \mathcal{D}(p(Z_{t+1}) \parallel Z_\theta) = \mathcal{D}(\mathbb{E}_\mathcal{T} p(y \mid s') \parallel Z_\theta)$.

* we are free to choose $\mathcal{D}$!

$$\mathrm{KL}(p(Z_{t+1}) \parallel Z_\theta) = -\int p(Z_{t+1}) \log Z_\theta + C = -\int \mathbb{E}_\mathcal{T} p(y \mid s') \log Z_\theta + C =$$

$$= -\mathbb{E}_\mathcal{T} \int p(y \mid s') \log Z_\theta + C = \mathbb{E}_\mathcal{T} \, \mathrm{KL}(p(y \mid s') \parallel Z_\theta) + C$$

✓ this can be evaluated through Monte-Carlo!

KL-**magic**

Our loss is $L = \mathcal{D}(p(Z_{t+1}) \parallel Z_\theta) = \mathcal{D}(\mathbb{E}_\mathcal{T} p(y \mid s') \parallel Z_\theta)$.

* we are free to choose $\mathcal{D}$!

$$\mathrm{KL}(p(Z_{t+1}) \parallel Z_\theta) = -\int p(Z_{t+1}) \log Z_\theta + C = -\int \mathbb{E}_\mathcal{T} p(y \mid s') \log Z_\theta + C =$$

$$= -\mathbb{E}_\mathcal{T} \int p(y \mid s') \log Z_\theta + C = \mathbb{E}_\mathcal{T} \, \mathrm{KL}(p(y \mid s') \parallel Z_\theta) + C$$

✓ this can be evaluated through Monte-Carlo!

× trick doesn't work for other divergences!

KL-**magic**

Our loss is $L = \mathcal{D}(p(Z_{t+1}) \parallel Z_\theta) = \mathcal{D}(\mathbb{E}_\mathcal{T} p(y \mid s') \parallel Z_\theta)$.

* we are free to choose $\mathcal{D}$!

$$\mathrm{KL}(p(Z_{t+1}) \parallel Z_\theta) = -\int p(Z_{t+1}) \log Z_\theta + C = -\int \mathbb{E}_\mathcal{T} p(y \mid s') \log Z_\theta + C =$$

$$= -\mathbb{E}_\mathcal{T} \int p(y \mid s') \log Z_\theta + C = \mathbb{E}_\mathcal{T}\, \mathrm{KL}(p(y \mid s') \parallel Z_\theta) + C$$

✓ this can be evaluated through Monte-Carlo!

× trick doesn't work for other divergences!

* KL requires Z_{t+1} and Z_θ share domain.

Distribution family

Options:

- Gaussian mixture
- Discrete

Distribution family

Options:

- Gaussian mixture
- Discrete ✓
  * KL-divergence of two discrete distributions is a simple cross-entropy

Distribution family

Options:

- Gaussian mixture
- Discrete ✓
    * KL-divergence of two discrete distributions is a simple cross-entropy

Let's $\mathcal{P}$ be a family of categorical distribution on the grid from V_{min} to V_{max} with N atoms (outcomes).

Distribution family

Options:

- Gaussian mixture
- Discrete ✓
  * KL-divergence of two discrete distributions is a simple cross-entropy

Let's $\mathcal{P}$ be a family of categorical distribution on the grid from V_{min} to V_{max} with N atoms (outcomes).

Parametrization:
For each action our neural network $Z(s, a)$ outputs N numbers, summing into 1

Calculating target

Suppose you have transition $(s, a, r, s', done)$, $Z(s, a) \in \mathcal{P}$. Then:

$$y(s') = r + \gamma Z(s', \max_{a'} \mathbb{E} Z(s', a'))$$

Calculating target

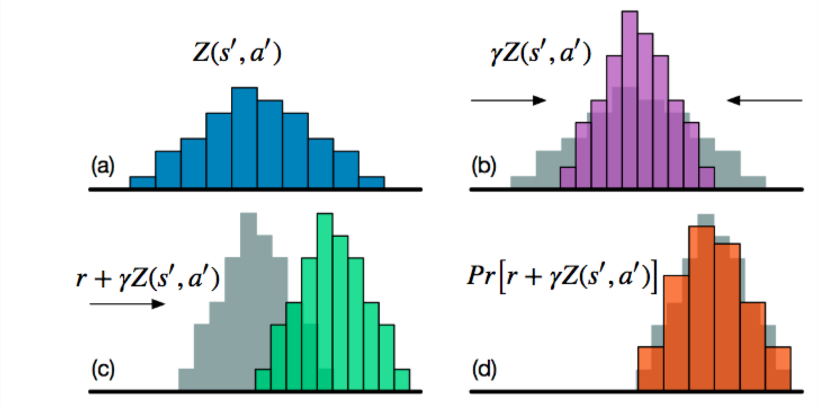
Suppose you have transition $(s, a, r, s', done)$, $Z(s, a) \in \mathcal{P}$. Then:

$$y(s') = r + \gamma Z(s', \max_{a'} \mathbb{E} Z(s', a')) \in \mathcal{P}?$$

Calculating target

Suppose you have transition $(s, a, r, s', done)$, $Z(s,a) \in \mathcal{P}$. Then:

$$y(s') = Pr\left[r + \gamma Z(s', \max_{a'} \mathbb{E} Z(s', a'))\right] \in \mathcal{P}$$



How it looks like

Failed to insert video into beamer ;o)

Rainbow DQN (2018)

Blend them all!

Multistep DQN: Motivation

Recall our target in classic DQN:

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

If we have nonzero reward at the end of M-step game, we need at least M iterations of Q-learning to «propagate» this reward to all visited states.

Multistep DQN: Motivation

Recall our target in classic DQN:

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

If we have nonzero reward at the end of M-step game, we need at least M iterations of Q-learning to «propagate» this reward to all visited states.

> Look more than one step ahead!

Multistep DQN: Realisation

- work with transitions $(s, a, r, r', r'', \ldots, r^{(M-1)}, s^{(M)}, done)$

Multistep DQN: Realisation

- work with transitions $(s, a, r, r', r'', \ldots, r^{(M-1)}, s^{(M)}, done)$
- transition reward $R = r + \gamma r' + \gamma^2 r'' + \cdots + \gamma^{M-1} r^{M-1}$

Multistep DQN: Realisation

- work with transitions $(s, a, r, r', r'', \ldots, r^{(M-1)}, s^{(M)}, done)$
- transition reward $R = r + \gamma r' + \gamma^2 r'' + \cdots + \gamma^{M-1} r^{M-1}$
- use new target:

$$y = R + \gamma^M \max_{a^{(M)}} Q(s^{(M)}, a^{(M)})$$

Multistep DQN: Realisation

- work with transitions $(s, a, r, r', r'', \ldots, r^{(M-1)}, s^{(M)}, done)$
- transition reward $R = r + \gamma r' + \gamma^2 r'' + \cdots + \gamma^{M-1} r^{M-1}$
- use new target:

$$y = R + \gamma^M \max_{a^{(M)}} Q(s^{(M)}, a^{(M)})$$

$\times$ the further we look the worser y approximates $Q^{\pi^*}(s, a)$

$\Rightarrow$ number of steps should be chosen carefully.

Multistep Categorical DQN

Recall categorical DQN target:

$$y = \Pr\left[r + \gamma Z(s', \underset{a'}{argmax}\, \mathbb{E} Z(s', a'))\right]$$

Multistep Categorical DQN

Recall categorical DQN target:

$$y = \Pr\left[r + \gamma Z(s', \underset{a'}{argmax}\, \mathbb{E}Z(s', a'))\right]$$

It can also be made multi-step:

$$y = \Pr\left[R + \gamma^M Z(s^{(M)}, \underset{a^{(M)}}{argmax}\, \mathbb{E}Z(s^{(M)}, a^{(M)}))\right]$$

Multistep Categorical DQN

Recall categorical DQN target:

$$y = \Pr\left[r + \gamma Z(s', \underset{a'}{argmax}\, \mathbb{E}Z(s', a'))\right]$$

It can also be made multi-step:

$$y = \Pr\left[R + \gamma^M Z(s^{(M)}, \underset{a^{(M)}}{argmax}\, \mathbb{E}Z(s^{(M)}, a^{(M)}))\right]$$

Loss stays the same:

$$L = \mathrm{KL}(p(y) \parallel p(Z))$$

Dueling Categorical DQN

Recall dueling DQN:

$$Q(s, a) = V(s) + A(s, a) - \underset{a}{mean}\, A(s, a)$$

Dueling Categorical DQN

Recall dueling DQN:

$$Q(s, a) = V(s) + A(s, a) - \underset{a}{mean}\, A(s, a)$$

Let's make our $Z(s, a)$ (modeling categorical distribution with N atoms) in dueling way:

$$Z(s, a) = V_N(s) + A_N(s, a) - \underset{a}{mean}\, A_N(s, a)$$

where $V_N(s)$ and $A_N(s, a)$ are categorical N-atomed distributions.

Dueling Categorical DQN

Recall dueling DQN:

$$Q(s,a) = V(s) + A(s,a) - \underset{a}{mean}\, A(s,a)$$

Let's make our $Z(s,a)$ (modeling categorical distribution with N atoms) in dueling way:

$$Z(s,a) = \ softmax(V_N(s) + A_N(s,a) - \underset{a}{mean}\, A_N(s,a))$$

where $V_N(s)$ and $A_N(s,a)$ are arbitrary N numbers[14].

[14] why couldn't we only add softmax?

Double dueling multi-step noised categorical DQN with prioritized replay AKA Rainbow

Rainbow: Combining Improvements in Deep Reinforcement Learning (2018):

Dueling + Multistep + Categorical + DQN +

Double dueling multi-step noised categorical DQN with prioritized replay AKA Rainbow

Rainbow: Combining Improvements in Deep Reinforcement Learning (2018):

Dueling + Multistep + Categorical + DQN +

- **Double:** use target network to evaluate

$$y = R + \gamma^N Z^{\text{target}}(s^{(N)}, \underset{a^{(N)}}{argmax}\, Z(s^{(N)}, a^{(N)}))$$

Double dueling multi-step noised categorical DQN with prioritized replay AKA Rainbow

Rainbow: Combining Improvements in Deep Reinforcement Learning (2018):

Dueling + Multistep + Categorical + DQN +

- **Double:** use target network to evaluate

$$y = R + \gamma^N Z^{\text{target}}(s^{(N)}, \underset{a^{(N)}}{argmax}\, Z(s^{(N)}, a^{(N)}))$$

- **Noisy:** add noise to all fully connected layers

Double dueling multi-step noised categorical DQN with prioritized replay AKA Rainbow

Rainbow: Combining Improvements in Deep Reinforcement Learning (2018):
Dueling + Multistep + Categorical + DQN +

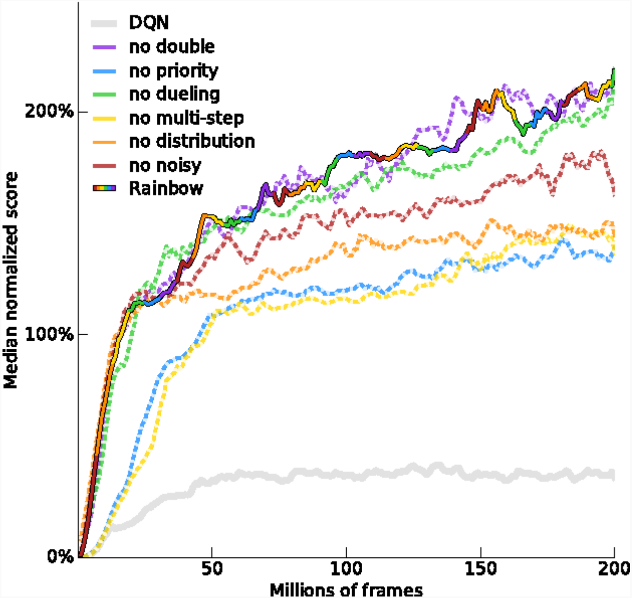
- **Double:** use target network to evaluate

$$y = R + \gamma^N Z^{\text{target}}(s^{(N)}, \underset{a^{(N)}}{argmax}\, Z(s^{(N)}, a^{(N)}))$$

- **Noisy:** add noise to all fully connected layers
- **Prioritized Replay:** just use it[15]

[15] guess proxy of transition priority

Do we really need all this?



Rainbow: resume

* all improvements are important as they address different problems

Rainbow: resume

* all improvements are important as they address different problems

$\times$ a lot of hyperparameters

Rainbow: resume

- * all improvements are important as they address different problems
- × a lot of hyperparameters
- ? Allegedly 10 hours for 7M frames on single GPU
  - :(I can't reproduce [16]

[16] 10 hours for 3M. Noise generation seems to be a problem!

NEXT: Going to Policy Gradients.