Lecture 13: Deep Reinforcement Learning

Matteo Hessel

UCL 2020

Updates

- Working on grading the first assignment,
- Some more time needed to finalise grades/feedback on individual questions,

Recap

- Deep learning and function approximation
  - Autodifferentiation
  - The JAX library
- Deep learning aware reinforcement learning
  - Recovering i.i.d updates and batching (experience replay, dyna)

Recap

- The deadly triad in deep rl
  - Soft divergence and target networks
  - Multi-step updates
  - Prioritised sampling
- Reinforcement learning aware deep learning
  - Inductive biases for reinforcement learning: dueling networks
  - Revisiting deep learning assumptions: network capacity
  - Bootstrapping and generalisations: leakage propagation

Outline

- Learning about many thing
  - GVFs and UVFAs
  - Distributional RL
  - Trade-offs

Learning about many thing

- Many deep RL algorithms only optimise for a very narrow objective
  - Narrow objectives induce narrow state representations,
  - Narrow representation can't support good generalisation,
  - Deadly triad, leakage propagation, ...
- Our agents should strive to build rich knowledge about the world
  - Learn about more than just the main task reward.

The reward hypothesis (Sutton and Barto 2018)

- All goals can be represented as maximization of a scalar reward,
  - All useful knowledge may be encoded as predictions about rewards
  - For instance in the form of ”general” value functions (GVFs),

General value functions (Sutton et al. 2011)

- A GVF is conditioned on more than just state and actions

  $$q_{c,\gamma,\pi}(s,a) = \mathbb{E}\left[C_{t+1} + \gamma_{t+1} C_{t+2} + \gamma_{t+1}\gamma_{t+2} C_{t+3} + \dots \mid S_t = s, A_{t+i} \sim \pi(S_{t+i})\right]$$

  where $C_t = c(S_t)$ and $\gamma_t = \gamma(S_t)$ where S_t could be the environment state
- $c : \mathcal{S} \to \mathbb{R}$ is the cumulant
  - Predict many things, including — but not limited to — reward
- $\gamma : \mathcal{S} \to \mathbb{R}$ is the discount or termination
  - Predict for different time horizons γ
- $\pi : \mathcal{S} \to \mathcal{A}$ is the target policy
  - Predict under many different (hypothetical) policies π

Example: Simple predictive questions

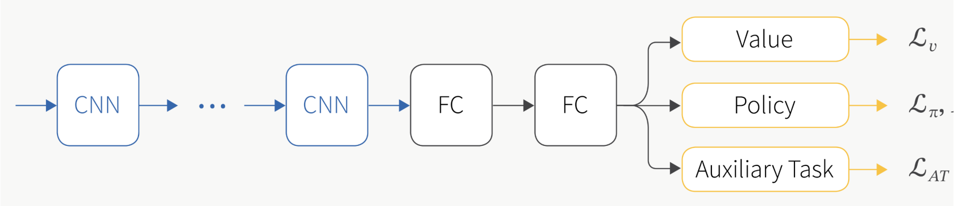
- Reward prediction:
  - $C_t = R_t$, $\gamma = 0$, under the agent's policy π
- Next state prediction:
  - $\{C_t^i = S_t^i\}_i$, $\gamma = 0$, under the agent's policy π
  - $\{C_t^i = \phi^I(S_t)\}_i$, $\gamma = 0$, under the agent's policy π

Predictive state representations (Littman et al. 2002)

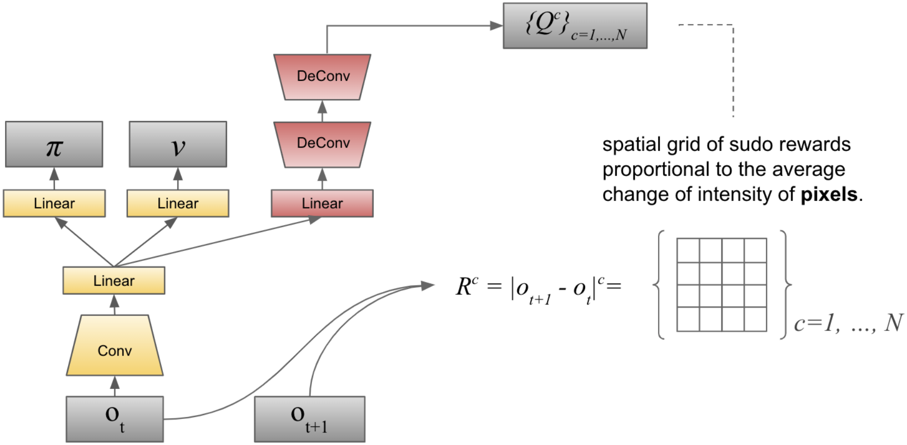
- A large diverse set of GVF predictions
  - will be a sufficient statistics for any other prediction,
  - including the value estimates for the main task reward.
- In predictive state representations (PSR):
  - Use the predictions themselves as representation of state,
  - Learn policy and values as a linear function of these predictions,

GVFs as auxiliary tasks (Jaderberg et al., 2016)

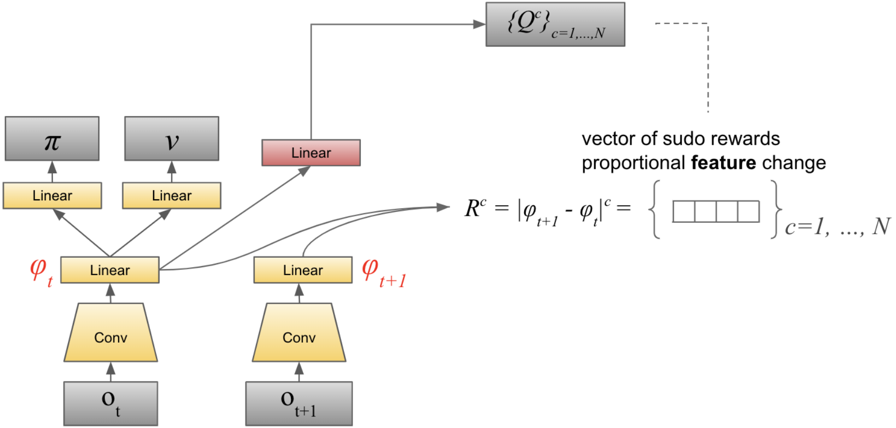
- GVFs can also be used as auxiliary tasks,
  - that share part of the neural network,
  - minimise jointly the losses for the main task reward and the auxiliary GVFs
  - force the shared hidden layers to be more robust,



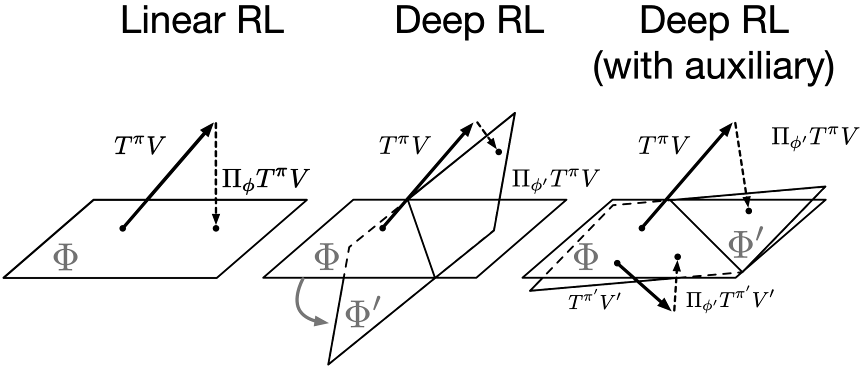
Example: Pixel Control (Jaderberg et al. 2016)



Example: Feature Control (Jaderberg et al. 2016)

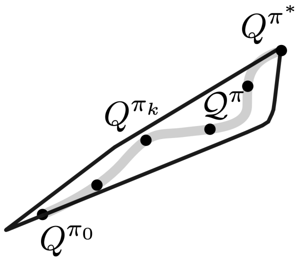


GVFs as auxiliary tasks



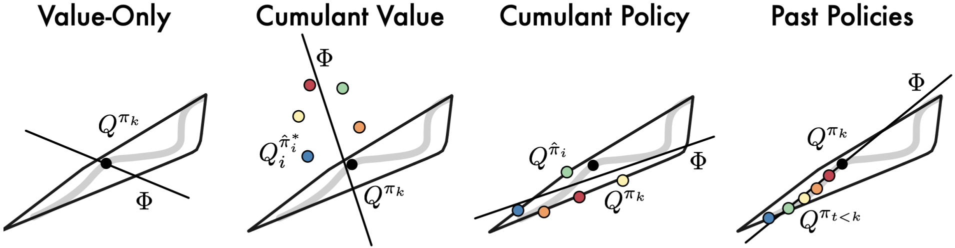
Value-Improvement Path (Dabny et al. 2020)

- Why regularise the representation?
  - Over the course of learning we need to approximate many value functions,
  - As we are tracking a continuously improving agent policy,
  - Must support all functions in the value improvement path from Q^{π_0} to Q^{π^*}



Value-Improvement Path (Dabny et al. 2020)

- Which GVFs best regularise the representation?



Discovery of useful questions (Veeriah et al. 2015)

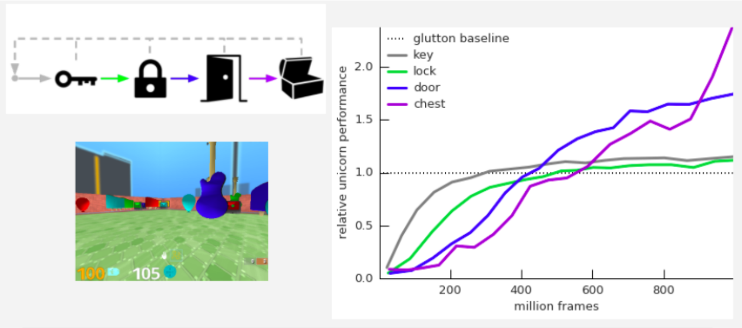
- Which GVFs best regularise the representation?
- The notion of value improvement path seeks a general answer,
- Instead, learn from experience what cumulants to predict,
- Using meta-learning (e.g. meta-gradients),

Learning about many things, and generalise (Schaul et al. 2015)

- GVFs-based auxiliary tasks help each other by sharing the representations,
- otherwise, each prediction is learnt independently
- Can we generalise what we learn about one GVF to other GVFs?
- Idea: feed a representation of (c, γ) is as **input**
- Allows generalization across goals/tasks within an environment
- This kind of GVFs are referred to as universal value functions

Learning about many things, off-policy

- We can learn about one cumulant off policy
- from data generated by a policy that maximises a different cumulant



'Unicorn' (Mankowitz et al., 2018)
Learn about many things to learn to do the hard thing

Distributional reinforcement learning (Bellemare, Dabney, Munos 2017)

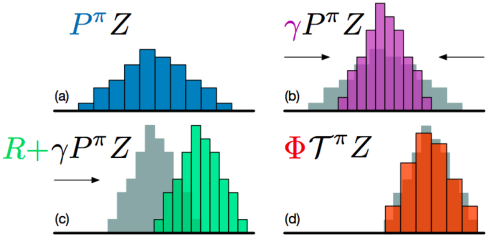
- Learning about many things may also mean learning the distribution of returns,
- rather than just approximating its expected value,
- Forces the representation (e.g., deep neural network) to be more robust,
- Knowing the distribution may be helpful for some things,
- For instance, you can perhaps reason about the probability of termination.

Distributional reinforcement learning

- A specific instance is Categorical DQN (Bellemare et al., 2017)
- Consider a 'comb' distribution on $\boldsymbol{z} = (-10, -9.9, \ldots, 9.9, 10)^\top$
- For each point of support, we assign a 'probability' $p^i_\theta(S_t, A_t)$
- The approximate distribution of the return s and a is the tuple $(\boldsymbol{z}, \boldsymbol{p}_\theta(s, a))$
- Our estimate of the expectation is: $\boldsymbol{z}^\top \boldsymbol{p}_\theta(s, a) \approx q(s, a)$ — use this to act
- Goal: learn these probabilities

Distributional reinforcement learning

1. Find max action:
   $$a^* = \underset{a}{\operatorname{argmax}}\ \boldsymbol{z}^\top \boldsymbol{p}_\theta(S_{t+1}, a)$$
   (use, e.g., θ^- for double Q)
2. Update support:
   $$\boldsymbol{z}' = R_{t+1} + \gamma \boldsymbol{z}$$
3. Project distribution $(\boldsymbol{z}', \boldsymbol{p}_\theta(S_{t+1}, a^*))$ onto support $\boldsymbol{z}$
   $$d' = (\boldsymbol{z}, \boldsymbol{p}') = \Pi(\boldsymbol{z}', \boldsymbol{p}_\theta(S_{t+1}, a^*))$$
   where Π denotes projection
4. Minimize divergence
   $$\mathrm{KL}(d' \| d) = -\sum_i p'_i \frac{\log p'_i}{\log p^i_\theta(S_t, A_t)}$$



Bottom-right: target distribution
$\Pi(R_{t+1} + \gamma \boldsymbol{z}, \boldsymbol{p}_\theta(S_{t+1}, a^*))$
Update $\boldsymbol{p}_\theta(S_t, A_t)$ towards this

Distributional reinforcement learning

- Categorical DQN is just one example of distributional reinforcement learning,
- There are many ways of modelling return distributions,
- For instance, we can model the quantiles via a GVF!
- ... see for instance (Dabney et al, 2018)

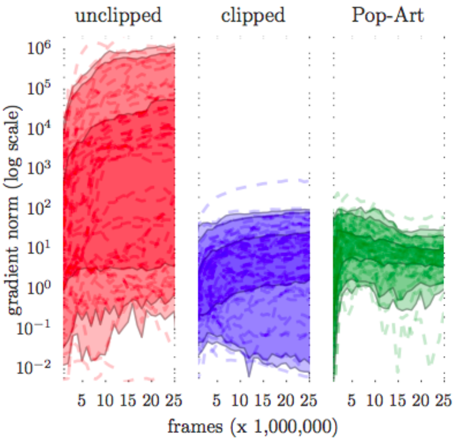
Learning about many things: trade-offs

- We want to learn as much as possible about the world,
- We only have limited resources (e.g. capacity, computation, ...),
- Different tasks compete for these resources,
- How do we trade-off between competing needs?

Learning about many things: trade-offs

- There is always a trade-off,
- Even if you don't do anything explicit
- The magnitude of the updates to shared weight differs across tasks,
  - e.g. scales linearly with the frequency and size of per-task rewards,
- Different tasks contribute to different degrees to representation learning,

Gradient norms



Update normalization

- In supervised learning we know before hand the dataset we will learn from,
  - We can normalise inputs and targets so that they have appropriate scales
- Many deep learning models do not work well without this
- In reinforcement learning we do not access to the "full dataset"
  - The scale of the values we predict also changes over time.
- Solution: adaptive normalization of updates

Adaptive target normalization (van Hasselt et al. 2016)

1. Observe target, e.g., $T_{t+1} = R_{t+1} + \gamma \max_a q_\theta(S_{t+1}, a)$
2. Update normalization parameters:

$$\mu_{t+1} = \mu_t + \eta(T_{t+1} - \mu_t) \qquad \text{(first moment / mean)}$$
$$\nu_{t+1} = \nu_t + \eta(T_{t+1}^2 - \nu_t) \qquad \text{(second moment)}$$
$$\sigma_{t+1} = \nu_t - \mu_t^2 \qquad \text{(variance)}$$

   where η is a step size (e.g., $\eta = 0.001$)

3. Network outputs $\tilde{q}_\theta(s, a)$, update with

$$\Delta\theta_t \propto \left(\frac{T_{t+1} - \mu_{t+1}}{\sigma_{t+1}} - \tilde{q}_\theta(S_t, A_t) \right) \nabla_\theta \tilde{q}_\theta(S_t, A_t)$$

4. Recover unnormalized value: $q_\theta(s, a) = \sigma_t \tilde{q}_\theta(s, a) + \mu_t$ (used for bootstrapping)

Preserve outputs

- Naive implementation changes all outputs whenever we update the normalization
- This seems bad: we should avoid updating values of unrelated states
- We can avoid this. Typically:

$$\tilde{\boldsymbol{q}}_{\boldsymbol{W},\boldsymbol{b},\theta}(s) = \boldsymbol{W}\phi_\theta(s) + \boldsymbol{b} \,.$$

- Idea: define

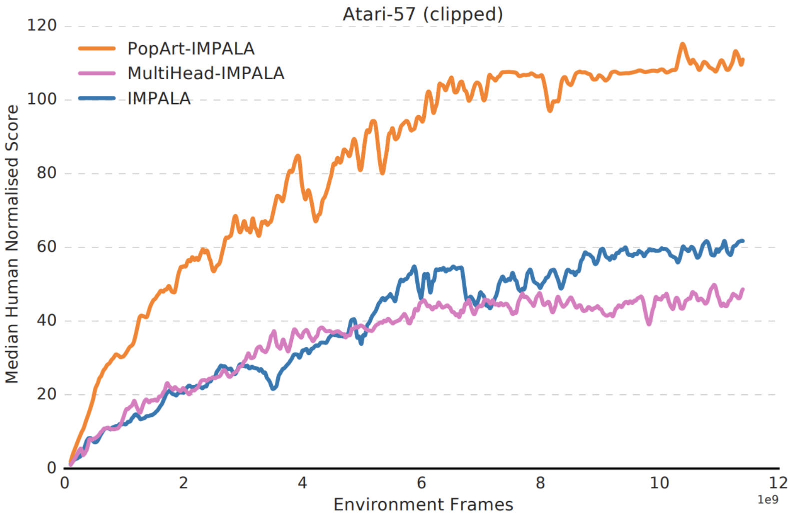
$$\boldsymbol{W}'_t = \frac{\sigma_t}{\sigma_{t+1}}\boldsymbol{W} \qquad\qquad \boldsymbol{b}'_t = \frac{\sigma_t \boldsymbol{b}_t + \mu_t - \mu_{t+1}}{\sigma_{t+1}}$$

  Then

$$\sigma_{t+1}\tilde{\boldsymbol{q}}_{\boldsymbol{W}'_t,\boldsymbol{b}'_t,\theta_t}(s) + \mu_{t+1} \quad = \quad \sigma_t\tilde{\boldsymbol{q}}_{\boldsymbol{W}_t,\boldsymbol{b}_t,\theta_t}(s) + \mu_t$$

- Then update $\boldsymbol{W}'_t$, $\boldsymbol{b}'_t$ and θ_t as normal (e.g., SGD)

Multi-task PopArt



Learning about many things: other benefits

- If we learn values and policies for multiple cumulants,
  - Temporally extended exploration,
  - Policy composition
- Value functions are not the only form of useful knowledge,
  - learn policy-independent environment models,
  - for instance, as auxiliary tasks
  - or to support model-based RL and planning,

Learning about many things: JAX

- There are open source JAX implementations for many of these ideas,
- See for instance the Rlax reinforcement learning package,
  - distributional value learning,
  - pixel-based cumulants,