Discovering Reinforcement Learning Algorithms:

https://arxiv.org/pdf/2007.08794.pdf

Paper Summary Notes

1 Main Ideas, High Level Assumptions

Question: can we *learn* a reinforcement learning algorithm?

- reinforcement learning algorithms are defined by rules that update an agent's parameters in order to optimize for expected cumulative rewards
- these rules are manually designed with theoretical and practical concerns over lots of research effort (e.g. $TD(\lambda)$, $A2C$, PPO)
- **assumption**: we can discover better general purpose update rules from data that can be applied across a variety of environments
- **approach**: jointly meta-learn *what to predict* (e.g. value functions) and *how to update the agent* using these predictions (e.g. bootstrapping) by interacting with a set of environments
- **novelty**: previous methods do not separate the agent and environment which makes the learned updates poor at generalizing to novel environments and architectures.

2 Approach and Formulation

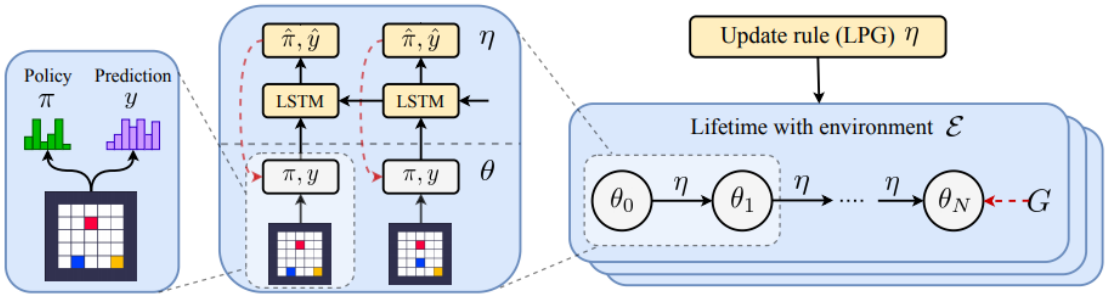
2.1 Objective

Learn a generalized update rule paramaterized by η, that maximizes the return over a distribution of environments and initial agent parameters

$$\eta^* = \arg\max_{\eta} \mathbb{E}_{\mathcal{E}\sim p(\mathcal{E})} \mathbb{E}_{\theta_0 \sim p(\theta_0)} [G]$$

- Distribution over environment: $p(\xi)$
- Distribution over agent initial parameters: $p(\theta_0)$
- Expected discounted future rewards: G
- sample the environment , and initialize a random agent. Then our *learned policy gradient* updates should lead this agent to optimal behaviour over time in expectation

2.2 Architecture



1. Agent produces policy output and prediction output:
   - meta parameters η require an agent paramaterized by θ to produce a policy $\pi_\theta(a|s)$, and a m-dimensional categorical feature vector $y_\theta(s) \in [0,1]^m$ as a function of the environment state s
2. Learned Policy Gradient:
   - The update rule (LPG) parameterised by η takes the agent outputs as input and unrolls an LSTM backward to produce targets for the agent outputs (π', y') from the trajectory of the agent
   - input at time t: $x_t = [r_t, d_t, y, \pi(a_t|s_t), y_\theta(s_t), y_\theta(s_{t+1})]$
   - output: π': policy update rule (specifies how the action probability should be adjusted), y': prediction update rule (specifies a target categorical distribution for a given state)
   - doesn't take as input the observation but rather the probability over the action, observation space so it's naturally invariant to these, making learned policy gradient able to generalize across environments
3. Update Rule:
   - agent is trained using meta update rule until end of episode, and meta-gradients are computed for meta parameters η
   - meta gradient is trained to maximize return after $K < N$ parameter updated by sliding window, where in each lifetime, a new environment and initial agent parameters are sampled

2.3 Gradient Updates

Agent Update (optimizing θ through updates induced by η)

$$\Delta\theta \propto \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)\hat{\pi} - \alpha_y \nabla_\theta D_{\mathrm{KL}}(y_\theta(s)\|\hat{y})\right]$$

- θ should maximize returns when updated using π' (direct effect on agents behaviour) which specifies how the action-probabilitity should be adjusted
- θ should minimize KL between agent target state (as computed by LPG), and categorical state prediction of the agent (no direct effect on agents behaviour until state is used in update)
- α_y is coefficient for the prediction update

Meta update (optimizing η)

$$\Delta\eta \propto \mathbb{E}_{\mathcal{E}}\mathbb{E}_{\theta_0}\left[\nabla_\eta \log \pi_{\theta_N}(a|s)G\right]$$

- LPG is meta-trained by taking into account how much it improves the performances of a population of agents interacting with different kinds of environments and running policy gradients to ascend on the above equation
- perform N agent updates using update rule η from initial parameters on a sampled environment and estimate policy grad for θ_N to find meta gradient that points in return of G. - in practice we truncate sliding window to $K < N$ updates because we have to backprop through entire trajectory for η (lots of memory)

Regularization

$$\mathbb{E}_{\mathcal{E}}\mathbb{E}_{\theta_0}\left[\nabla_\eta \log \pi_{\theta_N}(a|s)G + \beta_0 \nabla_\eta \mathcal{H}(\pi_{\theta_N}) + \beta_1 \nabla_\eta \mathcal{H}(y_{\theta_N}) - \beta_2 \nabla_\eta \|\hat{\pi}\|_2^2 - \beta_3 \nabla_\eta \|\hat{y}\|_2^2\right]$$

- to stabilize meta-gradients, we add regularization
- entropy regularize the meta outputs π', y' (prevents updates to update rules that are too deterministic)
- l2 regularizes the same (prevent aggressive updates to update rules)
- β_i are meta hyper-params

Balancing Hyperparameters

$$\eta^* = \arg\max_{\eta} \mathbb{E}_{\mathcal{E}\sim p(\mathcal{E})} \max_{\alpha} \mathbb{E}_{\theta_0 \sim p(\Theta)}\left[G\right]$$

- if we chose to fix agent hyperparams like learning rate then we are bound to fail when meta-training across a variety of environments,
- learning rate env A > learning rate env B $\implies$ $|\pi'_A|$ needs to be $< |\pi'_B|$. But since η is agnostic to environment, this would get contradicting meta-gradients.
- cannot prebalance hyperparmas since they depend on η (non-stationary)
- *solution*: modify our objective to the above which finds the optimal update rule 'given' the optimal hyperparameters (learning rate + coefficient α_y) for agent update)
- in practice we use bandit updates $p(\alpha|\xi)$ that sample hyperparameters for each lifetime and updates the sampling distribution according to the return at the end of each lifetime
- this makes meta gradient less noisy

2.4 PseudoCode

Algorithm 1 Meta-Training of Learned Policy Gradient

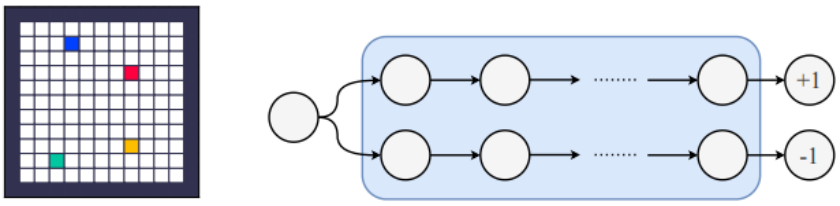
Input: $p(\mathcal{E})$: Environment distribution, $p(\theta_0)$: Initial agent parameter distribution
Initialise meta-parameters η and hyperparameter sampling distribution $p(\alpha|\mathcal{E})$
Sample batch of environment-agent-hyperparameters $\{\mathcal{E} \sim p(\mathcal{E}), \theta \sim p(\theta_0), \alpha \sim p(\alpha|\mathcal{E})\}_i$
repeat
  for all lifetimes $\{\mathcal{E}, \theta, \alpha\}_i$ **do**
    Update parameters θ using η and α for K times using Eq. (2)
    Compute meta-gradient using Eq. (4)
    if lifetime ended **then**
      Update hyperparameter sampling distribution $p(\alpha|\mathcal{E})$
      Reset lifetime $\mathcal{E} \sim p(\mathcal{E}), \theta \sim p(\theta_0), \alpha \sim p(\alpha|\mathcal{E})$
    end if
  end for
  Update meta-parameters η using the meta-gradients averaged over all lifetimes.
until η converges

3 Experiments

Research Questions:

1. How crucial is it to discover the semantics of the predictions?
2. What are the discovered semantics of predictions?
3. Can LPG learn predictions that lead to efficient bootstrapping?
4. How crucial is regularization and hyperparameter balancing?
5. Can LPG update rules learned for toy example be effective on complex atari games?

Training Environments



(a) Grid world (b) Delayed chain MDP

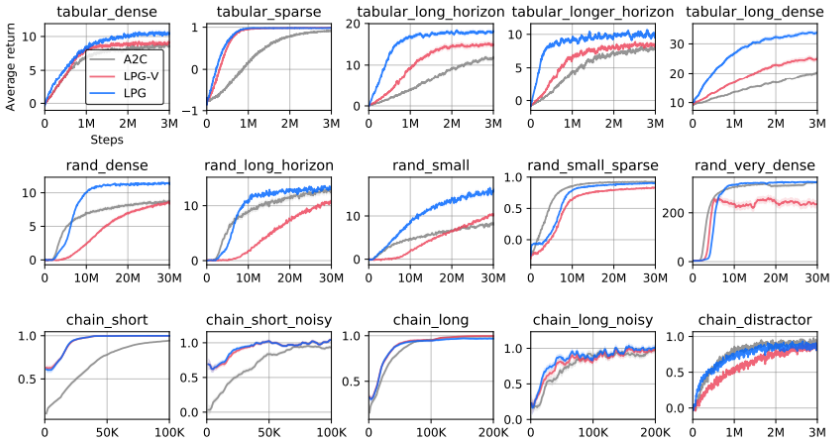
- training environments: aim to show delayed reward, noisy signals, sparse rewards
- tabular grid world (fixed spot objectives)
- random grid world (random spot objectives)

- delayed chain mdp (simple mdp with delayed rewards)

Baselines

- *LPG-V*: use value functions learned with $TD(\lambda)$ but learn update our own update rules (only learn π')
- *A2C* asynchronous actor critic state of the art

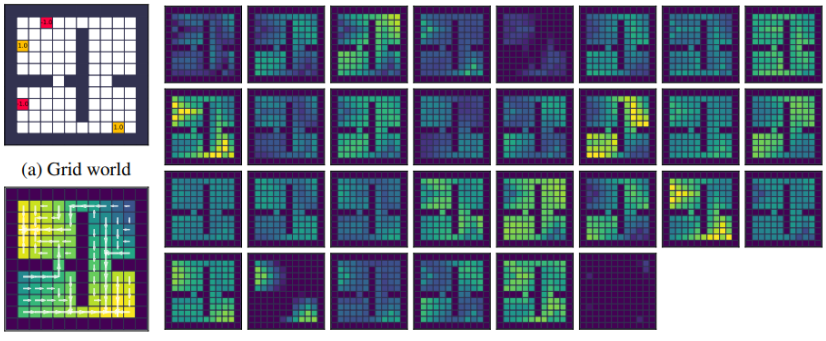
3.1 Experiment 1: Performance on Training Environments



- LPG does better than A2C, but LPG-V does not
- implies learning semantics (y') alongside updates is important to good results

Conclusion: it is important to learn semantics of predictions

3.2 Experiment 2: Analysis of Semantics (y')



(a) Grid world

(b) Policy and value

(c) Predictions (y)

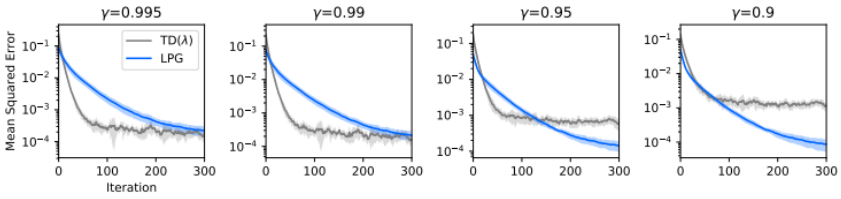
Figure 4: Visualisation of predictions. (a) A grid world with positive goals (yellow) and negative goals (red). (b) A near-optimal policy and its true values. (c) Visualisation of $y \in [0, 1]^{30}$ for the given policy in (b).

- Since the discovered semantics of prediction is the key for the performance, a natural question is what are the discovered concepts and how they work
- figure shows policy evaluation under a fixed policy using semantics y'

- the semantics resemble behaviour of a value function: some predictions have large values around positive rewarding states, and they are propagated to nearby states similarly to the true values in
- implicitly shows that the LPG is asking the agent to predict future rewards

Conclusion: learned semantics y' resemble value function behaviour

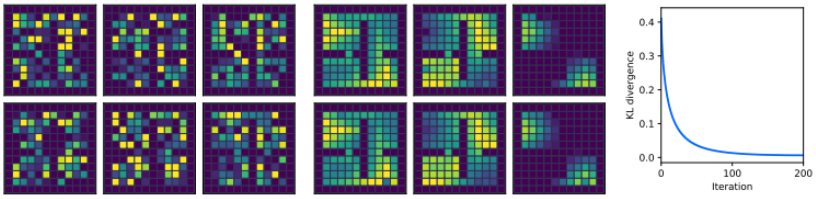
3.2.1 Value Regression



- train shallow neural net to predict true reward values from 30 dimensional y' values.
- Comparison against same model for under $TD(\lambda)$ shows similar ability which implies the values in y is rich enough to recover the value function

Conclusion: learned semantics y' can recover true state values

3.3 Experiment 3: Ergodic convergence

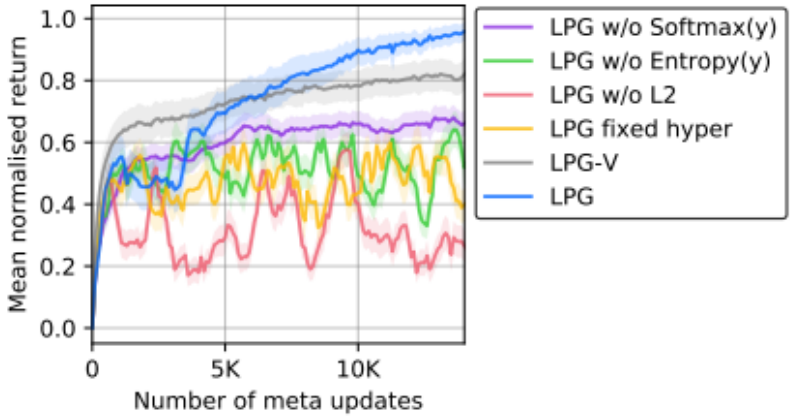


(a) Initial predictions (b) After convergence (c) KL divergence

- hand-crafted R.L. algorithms enjoy some convergence theory which guarantee's the ability to reach a (optimal) policy over time
- figure shows without any theoretical constraints, the LPG rules have stationary semantics (same predictions move to same next step)

Conclusion: LPG predictions can be used for bootstrapping

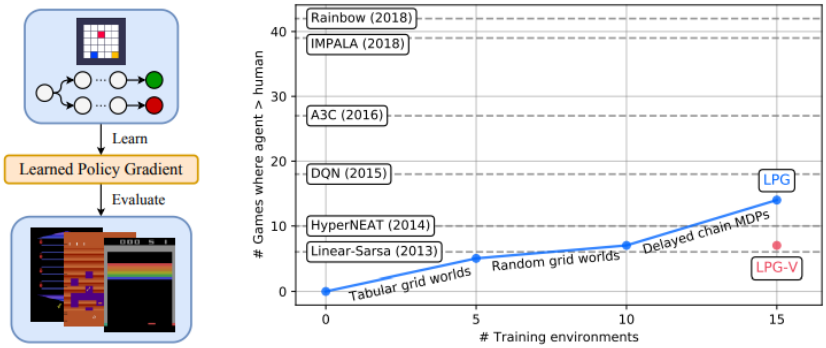
3.4 Experiment 4: Ablation Analysis



- tests how important the regularization and hyperparamater balancing is for the model

Conclusion: all these additions are necessary for stable LPG training

3.5 Experiment 5: Generalizing to Complex Environments



- meta paramaters are trained on toy environments to give good update rules on these simple envs as seen by experiment 1. Can these learned update rules do well on complex atari tasks?
- does reasonable close to hand-crafted advanced R.L. algorithms and even outperforms on some games
- generalizing improves with more variety in train environments

Conclusion: all these additions are necessary for stable LPG training

4 Practical Applications, Limitations

4.1 Applications

- potential to dramatically accelerate the process of discovering new reinforcement learning algorithms by automating the process of discovery in a data-driven way.

- shift the research paradigm from manually developing RL algorithms to building a proper set of environments so that the resulting algorithm is efficient.

4.2 Limitations

- is the update rule biased towards a specific reward function design?
- how expressive is the set of update rules we optimize over? It seems like we are agent update is a learning a policy gradient baseline