Encoding Human Domain Knowledge to Warm Start Reinforcement Learning

Andrew Silva, Matthew Gombolay

Georgia Institute of Technology

19th Oct, 2020

Motivation

- (Deep) RL disregards logical structure present in many domains.
- Knowledge from human experts can also be leveraged.
- Such knowledge can be encoded as propositional rules which can be used to *warm start* the learning.
- To bypass early random exploration and expedite learning.
- Related to IL and human-in-the-loop learning: usually require large labeled dataset.
- High level if-then checks are usually possible from a human.

Introduction: Propositional Logic Nets

- ProLoNets: Represent domain knowledge as propositional rules and encode them in a NN.
- Directly translates human expertise to RL agent's policy and begins learning immediately, sidestepping the IL and labeling phase.
- Use decision tree policies from humans to directly initialize a NN.
- Leverages readily available domain knowledge (from humans) while still retaining the ability to learn and improve over time using PG updates.
- Can also be used by untrained humas to provide the initial decision tree based policy.

ProLoNet Workflow Example

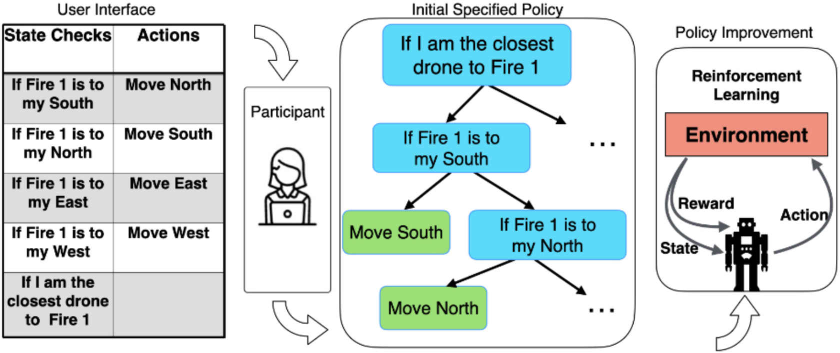


Figure 1: Humans interact with a UI of state-checks and actions to construct a decision tree policy that is then used to directly initialize a ProLoNet architecture and parameters. The ProLoNet can then begin RL in the given domain, outgrowing its original specification.

ProLoNet Initialization

- To intelligently initialize a ProLoNet, a human first provides a policy in the form of some hierarchical set of decisions (decision diagram).
- The human decisions are then translated into a set of weights $\vec{w}_n \in W$ and $\vec{c}_n \in C$.
- Each $\vec{w}_n$ determines which input feature(s) to consider and $\vec{c}_n$ is used as a threshold for the weighted features.
- Each decision node, D_n in the network is represented as $D_n = \sigma[\alpha(\vec{w}_n^T \vec{X} - c_n)]$.

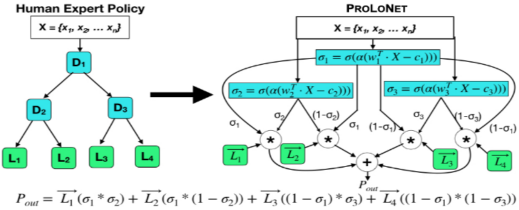


Figure 2: A traditional decision tree and a ProLoNet. Decision nodes become linear layers, leaves become action weights, and the final output is a sum of the leaves weighted by path probabilities.

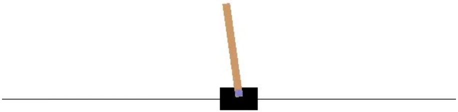
ProLoNet Initialization

Algorithm 1 Intelligent Initialization

Input: Expert Propositional Rules R_d
Input: Input Size I_S, Output Size O_S
$W, C, L = \{\}$
for $r \in R_d$ **do**
    if r is a state check **then**
        $\mathbf{s}$ = feature index in r
        $w = \vec{0}^{I_S}, w[\mathbf{s}] = 1$
        c = comparison value in r
        $W = W \cup w, C = C \cup c$
    end if
    if r is an action **then**
        $\mathbf{a}$ = action index in r
        $l = \vec{0}^{O_S}, l[\mathbf{a}] = 1$
        $L = L \cup l$
    end if
end for
Return: W, C, L

Example: Cart pole

- Knowledge solicited from a human: "If the cart's *x_position* is right of center, move left; otherwise, move right," and that the user indicates *x_position* is the first input feature of four and that the center is at 0.
- Initialize the primary node D_0 with $\vec{w}_0 = [1, 0, 0, 0]$ and $c_0 = 0$, following lines 5-8 in Alg. 1.
- Following lines 11-13, we create a new leaf $\vec{l}_0 = [1, 0]$ (Move Left) and a new leaf $\vec{l}_1 = [0, 1]$ (Move Right).
- Finally, we set the paths $Z(\vec{l}_0) = D_0$ and $Z(\vec{l}_1) = \neg D_0$. The resulting probability distribution over the agent's actions is a softmax over $(D_0\vec{l}_0 + (1 - D_0)\vec{l}_1)$.



Inference

- D_n: Likelihood of that condition being true. Similarly, $1 - D_n$: likelihood of being false.
- The network then multiplies out the probabilities for different paths to all leaf nodes.
- Every leaf $\vec{l} \in L$ contains a path $z \in Z$, a set of decision nodes which should be true or false in order to reach $\vec{l}$ as well as prior set of weights for each action $a \in \vec{a}$. E.g., in figure 2, $z_1 = D_1 * D_2$ and $z_3 = (1 - D_1) * D_3$.
- The likelihood of each action a in leaf $\vec{l_i}$ is determined by multiplying the probability of reaching leaf $\vec{l_i}$ by the prior weight of the outputs within leaf $\vec{l_i}$.
- Outputs of leaves are summed and passed through a softmax function to provide the final output distribution.

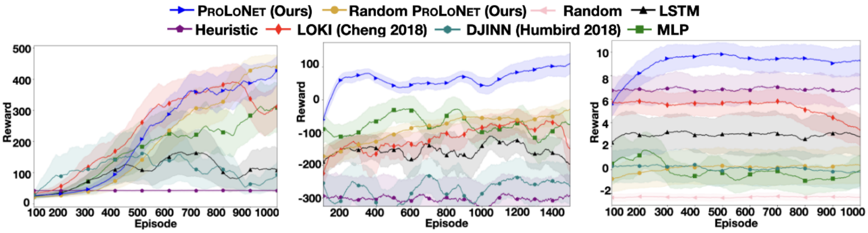
Example

- Consider an example cartpole state $X = [2, 1, 0, 3]$ passed to the ProLoNet from the previous example.
- For D_0, the network arrives at $\sigma([1, 0, 0, 0] * [2, 1, 0, 3] - 0) = 0.88$, meaning *mostly* true.
- This probability propagates to the two leaf nodes, making the network output $[0.88, 0.12]$.

Dynamic Growth and Experiments

- Dynamic growth of ProLoNets to learn complex policies: Maintain 2 copies of the actor: Shallower and Deeper.
- Shallower: Unaltered, initialized version. Deeper: Leaf transformed into a randomly initialized decision node with 2 randomly initialized leaves. Complex policy but added uncertainty.
- Shallower network generates actions; off-policy update after each episode; Entropy of leaves of both the networks are compared for deciding to augment the deeper network.
- Experiments: Cartpole, Lunar Lander, StarCraft, Wildfire Tracking. Compared against MLP and LSTM agents of LOKI (IL based framework) and DJINN (learned decision tree).

Results



(a) Cart Pole (b) Lunar Lander (c) FindAndDefeatZerglings

Summary

- Encode human and domain knowledge into a NN, representing the knowledge as propositional rules (decision trees).
- Human knowledge can *warm start* RL and we can skip the initial random exploration and learn in environments that are too complex for randomly initialized agents.
- ProLoNets beat IL+RL on traditional architectures.
- Superior policies even if we solicit information from average participants (need not be experts).

Backup

Algorithm 3 PROLONET Forward Pass

Input: Input Data X, PROLONET P
for $d_n \in D \in P$ **do**
  $\sigma_n = \sigma[\alpha(\vec{w_n}^T * \vec{X} - c_n)]$
end for
$\vec{A}_{OUT}$ = Output Actions
for $\vec{l_i} \in L$ **do**
  Path to $\vec{l_i} = Z(L)$
  $z = 1$
  for $\sigma_i \in Z(L)$ **do**
    if σ_i should be $TRUE \in Z(L)$ **then**
      $z = z * \sigma_i$
    else
      $z = z * (1 - \sigma_i)$
    end if
  end for
  $\vec{A}_{OUT} = \vec{A}_{OUT} + \vec{l_i} * z$
end for
Return: $\vec{A}_{OUT}$

Backup

Algorithm 2 Dynamic Growth

Input: PROLONET P_d
Input: Deeper PROLONET P_{d+1}
Input: ϵ = minimum confidence
$H(\vec{l_i})$ = Entropy of leaf $\vec{l_i}$,
for $l_i \in L \in P_d$ **do**
    Calculate $H(l_i)$
    Calculate $H(l_{d1}), H(l_{d2})$ for leaves under l_i in P_{d+1}
    if $H(l_i) > (H(l_{d1}) + H(l_{d2}) + \epsilon)$ **then**
        Deepen P_d at l_i using l_{d1} and l_{d2}
        Deepen P_{d+1} at l_{d1} and l_{d2} randomly
    end if
end for