Description of Idea for Operator Selection, Mixing Both Expert Knowledge and Learning
Document UUID: 3eb28d00-0d06-4b26-828e-126026308900

David Bayani

Intended Receipiants and Preamble

This document is intended for receipt only by Dr. Stefan Mitsch[1] and, if he is interested, Dr. André Platzer[2]. It details ideas that the author (me, David Bayani[3]) has had regarding selection of operators in Fanoos. The method as described here is in the process of being implemented.

Overview

Here I provide details and an overview of how operators are selected in Fanoos. The approach taken allows for leveraging expert rules for selecting operators as well as learning which operator to take. It supports classic logic-based rules, as well as approaches more closely related to soft classifiers.

The basic idea is as follows:

- There are an indexed set of operators, S_O, which may be applied to a state
- There are an indexed set of selectors, S_s, which each produce a (probability) distribution over S_o. Each selectors takes in a variety of information, including the entire history of use in Fanoos, the current state being applied to, and potentially more (basically anything it can get its hands on is allowed). For the sake of simplicity, we will write this as $s_i(*)$ where $s_i \in S_s$ - the point here is that $*$ is used to represent the variety of arguements s_i may have.
- A process weighs and combines the distributions produced by members of S_s to produce a final distribution over S_O.
- The final distribution is sampled to produce the operator to use.[4] This sampling is done in a way to help promote a healthy balance between exploration and exploitation.
- After the operator is applied, the state is formed, and the user makes a request, the user request is tranlated into a numeric feedback.
- The numeric feedback is used to adjust the weights given to each member S_s, as well as (via the standard mechanisms of UCB algorithm) the sampling of the final distribution.
- This process repeats until the user exits.

We list now some of the high-level ideas that have shaped this design:

Generally, there are three things that are available which we would like to leverage:

- (A1) number of times each operator was tried - we want to make sure we explore sufficiently
- (A2) success rate - we want to try and pick operators that work well, for our notion of what "working well" is.
- (A3) distance - we have a state which we want to apply and operator to, and we have all the prior states, operators applied to them, and results from the past. We would like to leverage knowledge of previous state's structure to decide what to do, as oppossed to simply choose operators based on what tends to work well when averaged across all states. As such, we would like to use the knowledge of how far different a state is from each prior states to help guide decisions for which operator is best for the current state.

Ideally, the approach could allow for both of the following to be worked-in effectively:

- (B1) learning of which operator to apply
- (B2) expert knowledge and guidance on which operator to use

The approach taken addresses each of (A1)-(A3) and (B1), (B2) above.

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1] https://orcid.org/0000-0002-3194-9759 , http://www.scopus.com/inward/authorDetails.url?authorID=23668291300&partnerID=MN8TOARS

[2] https://orcid.org/0000-0001-7238-5710 , http://www.scopus.com/inward/authorDetails.url?authorID=23393619700&partnerID=MN8TOARS , http://www.researcherid.com/rid/J-2507-2014

[3] https://github.com/DBay-ani

[4] This is likely to be change later. Both from prior experience and somewhat intuitively, while a randomized approach theoretically gives better results (classic one-arm bandit algorithms, and randomized weighted-majority for instance), in the case of user-interaction systems, they tend to require more data than is practical to start getting reasonable results; the user is only willing to put-up with so much.

Inference

This section provides details on how operators are selected; that is, how inference is performed.

Let $w_i \in \mathbb{R}$ be the weight that selector $s_i \in S_s$ is given by the system ; note that this value may be negative. We form the distribution to sample from as :

$$D_{sample}(*, \vec{w}) = \frac{(\sum_{i \in [|S_s|]} w_i s_i(*)) - min_i(\mathbb{1}(w_i \leq 0) w_i)}{(\sum_{i \in [|S_s|]} w_i) - min_i(\mathbb{1}(w_i \leq 0) w_i)}$$

The Upper Confidence Bound (UCB) algorithm, widely used from one-arm bandits, is then applied to $D_{sample}(*, \vec{w})$ as though each member of $D_{sample}(*, \vec{w})$ gave a success rate; this algorithm is responcible for producing the index of the operator to use. Notice that by using the (UCB) alogithm, we address (A1) while potentially respecting (A2) and (A3). That is, the selectors themselves will address (A2) and (A3) when forming the distributions, while we address (A1) at the very end by using the UCB algorithm to encorporate the selector suggestions with necessary exploration.

Learning

We will break our discussion of learning from feedback into two parts: how the selector-weights are updated given a reward signal, and how the reward signal is derived from user feedback.

Updating Selector Weights Given the Reward Signal

Suppose we are given a reward signal $y_t \in \{-1, 1, 0\}$ at iteration $t \in \mathbb{N} \backslash \{0\}$- in the next section, this signal is derived from user feedback and the history, but for now we suppose we have already derived it. Let O_t be the operator that was actually selected for iteration t (i.e., the one chosen at the end of inference). Tweaking notion slighty to encorporate a time-index, we update the weight of each selector as follows:

$$w_{i,t+1} = w_{i,t} + y_t(\ \ (s_i(*_t))(O_t) - U)$$

here, U is the probability that O_t would have been chosen at random (e.g., given a uniform distribution over S_o), and is simply $\frac{1}{|S_o|}$. The subtraction of u is nice for a few reasons. First, it provides the intuition that we positively (respectively, negatively) reward selectors that gambled more than random on an operator that ended up being correct (respectively, incorrect). Second, it helps prevent "coasting" on the success of other operators. Suppose that the collection of operators, as a whole, tend to be correct 90% of the time, but each individualy operator is only correct roughly 60% of the time (having probability mass on useful operators only 60% of the time, and 40% of the time having the majority of the probability mass on poor choices). Further, suppose that one of the selectors is simply a uniform distribution over all operators. The uniform-distribution selector may well eventually get higher weight than any of the selectors that actually provide meaningful guidance. This in itself is not a problem, since a selector that is always uniform would preserve the rankings of operator-liklihood [5], but if this uniform-selector occassionally outputs a non-uniform distribution, the final distribution would unduely be swayed by it. Note that reducing the weight for selectors that are wrong does not itself resolve this "issue" [6], since if the uniform-selector "waits" long enough, it will recuporate its weight from betting on an poor operator.

A concern about this approach relates to the proposal to randomize selection. While in principle this is good and appropraite for multiple reasons, in a user-interaction system that presumably has "little"[7] data to tune with, the amount of variance may be unacceptable. This is certainly the case from my experience working on RedThread (Rabbany, Bayani, and Dubrawski 2018), where randomized approaches did notably worse. Here, if randomization ultimately proves to be unacceptable, I will do as I did for RedThread, and simply sample the operator with highest weight. If randomization is abandoned, formal justification may still be avaiable by leveraging ideas from the Majority Vote or Weighted Majority Vote algorithm (Littlestone and Warmuth 1994) [8] (as opposed to the Randomized Weighted Majority Vote algorithm).

As a closing note, it is worth considering further whether this update rule has any connection to, or can leverage lessons learned from, variance reduction via baselines in vanilla policy gradients (e.g., (Williams 1992)). Notice that: (1) the "shape" and, in some sense, "content" of the proposed update are not too dissimilar in appear compared to the vanilla policy gradient, (2) we clearly have a baseline term (U), and (3) like in PG, we sample a final distribution to determine what action we take. The approach proposed in this write-up was constructed prior to making this sort of connection, but it is no surprise similar forms might appear in similar problem settings - it is just not clear whether the relation has more to offer than a superficial one, or if it is about as useful as citing the entire sub-field of model-free RL as being relavent.

Producing the Reward Signal

We now discuss how, in this initial implementation, we plan to form the y_t used in the previous subsection. Below, for ease of discussion, we will use "iteration t" and "time t" interchangably.

Suppose that our process in Fanoos overall follows the flow of:

- produce the state for time t.

[5] Well, strictly speaking there would be an issue even in this case, since a large, uniform influence would serve to "flatten" the distribution, making it more likely to select operators that otherwise would have little or no chance.

[6] One can argue to what degree this is actually a real "problem". Often in ML we accept occassional/rare missteps so long as the typical case behaves appropraitely.

[7] By modern ML standards

[8] If interested in examining this work, I suggest looking up lecture material for a more streamlined and digestable presentation. For example https://www.cs.cmu.edu/~15451-s16/lectures/lec20.pdf

- Show responce for time t by extracting relavent information from the state at iteration t.
- Receive user-feedback for time t, f_t.
- Choose operator at iteration t.

Types of user feedback currently supported are listed in Table 1. From this flow, it is important to note that we have the user feedback for the state prior to applying the operator. Once we apply the operator to produce a new state and show it to the user, we will receive more user feedback, f_{t+1}. While many different methods for forming y_t can be supported, the initial implementation uses the reward function described in Table 2. The idea behind this current reward function is that we want to learn which operators do in fact produce more (respectively, less) abstract descriptions on user demand. We take the user "reversing" their request (e.g. $f_t = l$ and $f_{t+1} = m$) to be a sign that abstraction definetly went in the correct direction and to a non-trivial degree. Similarly, if $f_{t+1} = b$, we take it as a sign that we satisfied the user's previous request. Clearly there are potential issues with this reward function, particularly in cases where we "got close to the right abstraction level but overshot/undershot", but at the very least it seems to be a reasonable way to guide operator selection to handle the cases that are not so debatable.

Selectors to Be Tried

Having desribed how selectors are used, we now describe further the selectors themselves and some flavors of implementation.

First, we discuss what selectors in general are capable of. As stated earlier, these are the components in the process that can encorporate expert knowledge ((B1)), knowledge of how good operators are ((A2)), and how much the current state looks like previous states ((A3)). Further, since they output distributions over operators (that are ultimately decided via random sample), they support classical logic-based rules by putting all support on a subset of the operators, and naturally include a mechanism for breaking ties.

While all these capabilities and roles are well-and-good, ultimately some specific implementations of selector(s) need to be provided that can fill them. For an initial and reasonable implementation, we consider selectors based on KNN (and maybe KDE) with some human expert filtering. We will describe these "intelligent" selectors next, but first we note that, in order to encourage exploration, we always include a uniform-selector, which simply puts equal weight on all operators. For the rest of the selectors, we do the following:

- Let v_j be the value of field j in the state; that is, some obervable aspect of the state we know about. This j is fixed for the selector -each selector looks at exactly one field, but the same field may be considered by multiple selectors.
- Let k be a positive integer specified for the selector. Find the top k states in the history in order of descending distance from the current state, based on v_j (i.e., the distance between state A and state B is $|A.v_j - B.v_j|$).
- The distribution that the selector returns is the normalized success rate for each operator that was applied to one of the states from the prior bullet point to form a next-state (i.e, $\text{weight}(\text{operator}_i) = \frac{\text{success rate of operator}_i}{\sum_{j=0}^{|S_o|} \text{success rate of operator}_j}$). While many reasonable alternatives for weighing operators would encoporate information about how often an operator has been used in addition to its success rate, we simply leverage success rate here in part because the final distribution sampling encorporates usage information. It is worth noting, however, that the scheme proposed here would promote the use of operators that have been used less often and have been "lucky" (i.e., successful, but primarly due to chance), since the UCB algorithm used on the final distribution preferences operators used less often. Notice, however, that the symmetric case is not true - this approach does not overly disadvantage operators that have been used less and been "unlucky" thus far - the UCB algorithm only improves the chances of selection above the raw score.

Notice that the above description explicitly uses notitions of distance between states and operator success rates, covering both (A2) and (A3).

Other Reasonable Alternatives

A promising set of selectors to use instead of or in addition to the above are KDE (Kernel Density) based, with different bandwidth parameters instead of k. These approaches would leverage all the available states-operator data by weighing impact by the distance between states instead of uses a hard-cutoff to limit impact of more distant states. Before getting into the technical details, it is worth noting than an immedaite concern of this KDE approach is that there would not be enough data to train a KDE well. Often, simplier models (such as KNN) do noticably better than more sophisticated models (like KDEs) when the data to train them is relatively small - and since we are basing success on user feedback, it is not reasonable to assume a dataset that is, by ML standards, large.

The KDE approach would need to treat positive and negative labels (i.e., the y) in two different distributions or as a dimension of a distribution, then eventually combine the probabiliy-positive and probability-negative into one value. One reasonable but adhoc way to do this is to compute $\sqrt{(\text{probability-success})(1 - \text{probability-failure})}$. More principled methods of joining these values could borrow ideas from the likelihood ratio test.

Yet another alternative would be to use logistic regression. Features could be used to capture success and failure at certain distances from the current state[9], and use this to regress out a class liklihood; while this could encorporate more information than the KNN, it should be more data efficient than a KDE if we suppose a head-to-head match where both have the same amount of training data - this is due to logistic regression being a parametric method as oppossed to the non-parametric KDE. This approach, however, would

[9]For instance, $f_i = y(\text{distance})^{-1}$, where i stands for the feature of the i^th closest state

Types of User Feedback	
short-hand	description
l	make the description less abstract
m	make the description more abstract
h	allow the user to examine the history. For instance, view or return to prior states.
u	allow the user to specify predicates to disallow (forcing a difference description of a state).

Table 1: Description of the types of user-feedback Fanoos currently supports

f_t	f_{t+1}	y_t
m	m	-1
m	l	1
m	b	1
l	l	-1
l	m	1
l	b	1
m/l	h/u	0
h/u	(any)	0
b	(none)	(none)

Table 2: Description of the types of user-feedback Fanoos currently supports. Note that when the user enters "b" at iteration t, the line of question-and-answering breaks, so there is not feedback or reward for iteration t+1, since there is no iteration t+1.

not have quite as much data as a KDE available since, as I propose it here, it requires a window which may consume a number of test-points that is non-trivial when considering the "few" samples available for each operator in the interactive system. Variations on tones listed in this paragraph exist - for example, building off of exponential moving average where "time" is actually the state-distance - etc., etc. In general, there are a number of ways one can easily imagine to hook something up using basic parametric methods to attempt this task. Which of those possibilities is worth-while and effective in a "small" sample scenerio is a different story. It is worth stating that one of the appeals of a KNN and KDE approach is their simplicity and easy of implementation in the early development I am discussing here.

Incorporating Some Large-Grain Expert Knowledge

In the process of collecting data and producing a distribution over operators, we may use expert knowledge to limit which operators are candidates to have support[10]. In particular, there are many operators we have strong a priori believe will decrease (or respectively, increase) the granularity of the boxes produced in the reachability analysis, simply based on the refinement parameters they pass to the CEGAR-like analysis. If the user requests that the abstraction level be decreased, we may filter out operators that increase the granularity of the reachability hyper-cubes[11] ; the case where the user wants to increase abstraction is symmetric. This allows us to encorporate some basic knowledge into operator selection while enabling the learning of which specific operator should be chosen in each regime.

[10]Support - meaning positive probability mass put on those options.

[11]If desired, so to encourage exploration of operators we might not have thought to try, this filtering could be applied probabilistically, but likely doing that will wait until data shows the need for it. For operators whose circumstance-specific impact we have a good idea of, determinisitc filtering is almost certainly the best idea at this phase of development. The last thing we need is another hyper-parameter to tune.

References

Littlestone, N., and Warmuth, M. K. 1994. The weighted majority algorithm. *Information and computation* 108(2):212–261.

Rabbany, R.; Bayani, D.; and Dubrawski, A. 2018. Active search of connections for case building and combating human trafficking. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2120–2129. ACM.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, 229–256.