Deep RL and Tree-search for Feature Selection

Wendelin Böhmer (TU Delft), Rong Guo (TU Berlin)

February 16, 2022

Todo list

1 Domain Background

Background:

1. Pseudocode of AlphaZero.
2. Feature selection literature.

2 Problem/Solution Statement

MDP fomulation

Feature selection as a reinforcement learning problem (Gaudel and Sebag, 2010)

Given a feature set $\mathcal{X}$, the state space $\mathcal{S}$ of the MDP is the powerset of $\mathcal{X}$. We define a tree $\mathcal{T}$ where each node S correspond to a state. The action is to select a child node S_c of the parent node S, $\{S_c \subset \mathcal{X}, \exists\, \boldsymbol{x} \in \mathcal{X} \backslash \mathcal{S}, S_c = S \cup \{\boldsymbol{x}\}\}$

Goal: FS $\rightarrow$ generalization error. The features are used in the classification/regression model. Feature selection and model construction are combined in one end-to-end process.

- Training

$$\mathcal{L} = (z - v)^2 - \pi^T \log(P) + c\|\theta\|^2 \tag{1}$$

  The v, π guides the MCTS, while MCTS can be viewed as a way of building v, π (policy iteration).

- Test

Bellman optimality

1. Quantifiable: expressed in math/logic terms
2. Measurable: metric
3. Repliable

3 Dataset & Input $ Metric

1. Environment/Experiments.
2. Benchmark model & results, which contextualises existing methods/knowledge in the domain.
3. Metric: objectively compare the solution model with the benchmark model.

4 Theoretical workflow

4.1 End-to-end formalization

We are given a training set $\{\boldsymbol{x}^i\}_{i=1}^n \subset \mathbb{R}^{|\mathcal{A}|}$ of feature-vectors (e.g. MNIST images). Our task is to classify the corresponding labels $\{y^i\}_{i=1}^n$ (e.g. shown numbers $0 \dots 9$) by maximizing the probability $p_\theta(y^i|\boldsymbol{x}^i)$ parameterized by θ. However, the classification shall be performed with very few features, which we can select in an iterative process.

Starting with a fully hidden vector $\boldsymbol{x}_0^i$, a stochastic policy $\pi_\phi(a|\boldsymbol{x}_t^i)$, parameterized by ϕ, will select which feature $a \in \mathcal{A}$ will be revealed next. This implies a transition model $P(\boldsymbol{x}_{t+1}^i|\boldsymbol{x}_t^i, a)$, which reveals feature a of masked vector $\boldsymbol{x}_t^i$. The cross-entropy classification loss after t time steps is

$$\mathcal{L}_t := -\tfrac{1}{n}\sum_{i=1}^n \mathbb{E}_\pi\Big[\ln p_\theta(y^i|\boldsymbol{x}_t^i)\Big] \quad \text{where} \quad \mathbb{E}_\pi\Big[f(\boldsymbol{x}_t^i)\Big] := \mathbb{E}\Big[f(\boldsymbol{x}_t^i)\,\Big|\, \substack{a_k\sim\pi_\phi(\boldsymbol{x}_k^i) \\ \boldsymbol{x}_{k+1}^i\sim P(\boldsymbol{x}_k^i,a_k)}, 0 \le k < t\Big]\,. \tag{2}$$

We can optimizes all classifications after up to T reveals in one end-to-end loss functions:

$$\mathcal{L} := \sum_{t=1}^T \mathcal{L}_t = -\tfrac{1}{n}\sum_{i=1}^n \underbrace{\mathbb{E}_\pi\Big[\sum_{t=1}^T \overbrace{\ln p_\theta(y^i|\boldsymbol{x}_t^i)}^{r_{t-1}^i}\Big]}_{J^\pi}\,. \tag{3}$$

Note that this is also the objective J^π of reinforcement learning with rewards $r_t^i := \ln p_\theta(y^i|\boldsymbol{x}_{t+1}^i)$. We can therefore optimize the policy π_ϕ with the REINFORCE algorithm (Williams, 1992):

$$\nabla_\phi \mathcal{L} = -\tfrac{1}{n}\sum_{i=1}^n\sum_{t=1}^T \mathbb{E}_\pi\Big[\ln p_\theta(y^i|\boldsymbol{x}_t^i)\,\nabla_\phi \ln \pi_\phi(a_{t-1}|\boldsymbol{x}_{t-1}^i)\Big]\,. \tag{4}$$

To learn both models simultaneously end-2-end, we can therefore use the combined loss[1]:

$$\mathcal{L}^{\text{e2e}} := -\mathbb{E}_\pi\Big[\tfrac{1}{n}\sum_{i=1}^n\sum_{t=1}^T \Big(\ln p_\theta(y^i|\boldsymbol{x}_t^i) + \lambda \perp\big(\ln p_\theta(y^i|\boldsymbol{x}_t^i)\big)\,\ln \pi_\phi(a_{t-1}|\boldsymbol{x}_{t-1}^i)\Big)\Big]\,. \tag{5}$$

Note that the choice of λ is irrelevant, unless the two both models p_θ and π_ϕ share parameters. One could also use actor-critic algorithms (Sutton et al., 2000). However, the downside is that all samples have to be *on-policy*, that is, sampled by the current policy π_θ. This is slow and can lead to catastrophic forgetting (which can be counteracted by modern algorithms, Schulman et al., 2015, 2017, but this is another story).

4.2 Using tree search

Starting with AlphaGoZero (Silver et al., 2017), Deepmind's work on AlphaGo (Silver et al., 2016) has moved away from traditional reinforcement learning and uses in their latest iterations (Silver et al., 2018; Schrittwieser et al., 2020) a classification loss instead:

$$\mathcal{L}' := -\tfrac{1}{n}\sum_{i=1}^n\sum_{t=0}^{T-1} \mathbb{E}_{\mathcal{T}}\Big[\ln \pi_\phi(a_t|\boldsymbol{x}_t^i)\Big]\,, \tag{6}$$

where $\mathcal{T}(a|\boldsymbol{x}_t^i)$ denotes the tree-search operator starting at, or continuing from, $\boldsymbol{x}_t^i$. On the surface tree-search often yields an already good policy, which is then distilled into the policy π_ϕ. However, this way π_ϕ can never become better than $\mathcal{T}$. If not the entire search-tree can be computed efficiently, and the performance of $\mathcal{T}$ is therefore often suboptimal. Silver et al. (2017) argue that this changes fundamentally when we can use π_ϕ in $\mathcal{T}$ (now denoted $\mathcal{T}_\pi$). If we can guarantee that $\mathcal{T}_\pi$ is a *policy improvement operator*, i.e. $\mathcal{L}(\mathcal{T}_\pi) \ge \mathcal{L}(\pi_\phi)$, the learned policy will keep improving. This can be guaranteed by making sure that the action sequences

[1] Where $\perp$ stops the gradient flow, e.g. using `detach()` in `pytorch`.

that would be likely under π_ϕ are included in the tree search first, e.g. by implementing a preferential expansion heuristic (policy orders actions during node expansion) or by drawing the first m actions in any expanded node from the policy π_ϕ (e.g. by making π_ϕ a softmax).

Keep in mind that this does not allow to train the classifier with the same loss function, but one should still consider adjusting the classifier to the distribution induced by the learned policy π_ϕ, e.g., by minimizing $\mathcal{L}^{\text{ts}} := \mathcal{L} + \lambda\mathcal{L}'$.

4.3 On-policy learning for tree-search

Using tree-search $\mathcal{T}(a_t^i|\boldsymbol{x}_t^i)$ on individual samples $\boldsymbol{x}^i$ induces another problem that Cheng was pointing out in our discussion: different samples i can have different optimal actions a at time t. Training with actions found by a tree-search will therefore yield the distribution

$$\xi(a_t|\boldsymbol{x}_t) \quad := \quad \tfrac{1}{n}\sum_{i=1}^{n} \mathbb{I}[\mathcal{T}(a_t|\boldsymbol{x}_t^i) = 1]\,, \tag{7}$$

where $\mathbb{I}$ denotes the indicator operator, which is 1 if the argument is true and 0 otherwise. This induces two problems:

1. In difference to tree search, which always knows the current sample $\boldsymbol{x}^i$ and therefore which feature should be revealed, the classification loss $\mathcal{L}'$ uses the expectation $\mathbb{E}_\mathcal{T}$ to imitate ξ. This means the learned policy π will have to make some choices randomly when faced with unknown $\boldsymbol{x}_t$. These decisions will lead to states $\boldsymbol{x}_t^i$ that the decision tree would not have visited and which is the policy has therefore not been trained for.

2. The distribution $\xi(a_t|\boldsymbol{x}_t)$ may not be the optimal for an unknown partially revealed $\boldsymbol{x}_t$, as another action, that is suboptimal for each image, may reveal the most information to distinguish between samples and would therefore be optimal for an unknown $\boldsymbol{x}_t$.

4.3.1 On-policy learning

First, note that both problems appear due to tree-search. Minimizing the end-to-end loss $\mathcal{L}^{\text{e2e}}$ should learn to select actions that optimize the long term accuracy for all samples and do this *on-policy* by training on the rollouts of the policy itself. We can introduce the latter to tree-search by sampling $\boldsymbol{x}_t^i$ with the learned policy, but selecting the targets with $\mathcal{T}$:

$$\mathcal{L}'' \quad := \quad -\tfrac{1}{n}\sum_{i=1}^{n}\sum_{t=0}^{T-1} \mathbb{E}_\pi\Big[\ln \pi_\phi(\bar{a}_t|\boldsymbol{x}_t^i)\,\Big|\,\bar{a}_t \sim \mathcal{T}(\cdot|\boldsymbol{x}_t^i)\Big]\,. \tag{8}$$

Note that this objective will still imitate ξ, but on the entire distribution of samples reachable by π_ϕ. For tree-search algorithms that only output an optimal sequence of actions, this would require to rerun the search whenever the policy decides to alter that sequence. The same is advisable in regular intervals if the tree search only returns approximately optimal sequences (which have seen less data deeper in the tree).

4.3.2 Fixed tree search

The second problem above is harder to solve with tree-search. To demonstrate this, imagine some samples $\{\boldsymbol{x}^i\}_{i=1}^n$, which at some time $t > 0$ look the same, i.e. $\boldsymbol{x}_t = \boldsymbol{x}_t^i, \forall i$, but have different optimal actions $\bar{a}_t^i \sim \mathcal{T}(\cdot|\boldsymbol{x}_t^i)$ according to tree-search, because tree-search can look into the future where the samples differ from each other. However, there might be another action $\bar{a}_t^* \neq \bar{a}_t^i, \forall i$, which is sub-optimal for each sample but optimal for their combination:

$$p_\theta\big(y^i|P(\boldsymbol{x}_t^i, \bar{a}_t^i)\big) > p_\theta\big(y^i|P(\boldsymbol{x}_t^i, \bar{a}_t^*)\big)\,, \quad \text{but} \quad \sum_{j=1}^{n} p_\theta\big(y^j|P(\boldsymbol{x}_t^j, \bar{a}_t^i)\big) < \sum_{j=1}^{n} p_\theta\big(y^j|P(\boldsymbol{x}_t^j, \bar{a}_t^*)\big)\,, \quad \forall i\,. \tag{9}$$

For optimal classification of unknown samples $\boldsymbol{x}_t$, we should aim to predict $\bar{a}_t^*$, not $\bar{a}_t^i$. This sounds like the "joint tree search" procedure that Julia has implemented, but I would like to point out that this is only optimal when $\boldsymbol{x}_t = \boldsymbol{x}_t^i, \forall i$, that is, when there is no difference in the

observations. Instead, I would propose to average the tree search *only* over the samples that have the same observation as current sample $\boldsymbol{x}_t$, i.e. $\mathcal{X}(\boldsymbol{x}_t) := \big\{ i \,\big|\, \|\boldsymbol{x}_t^i - \boldsymbol{x}_t\| < \epsilon, 1 \le i \le n \big\}$.

It took me a while to come up with the value definition of the corresponding tree, so please check the following carefully. A node is characterized by the visible features $\boldsymbol{x}_t$ and has a list of children $\mathcal{C}_a$ for each explored action $a \in \mathcal{A}$. Those children $\boldsymbol{x}' \in \mathcal{C}_a$ are the result of revealing feature a in a sample $\boldsymbol{x}_t^i$ from the set $i \in \mathcal{X}(\boldsymbol{x}_t)$ that is consistent with $\boldsymbol{x}_t$. As multiple paths can lead to the same node $\boldsymbol{x}'$, children should be stored in a hash table to break transpositions. With every forward path choosing a in this node, a random (or enumerated, or heuristic) consistent sample $i \in \mathcal{X}(\boldsymbol{x}_t)$ is chosen and the successor $\boldsymbol{x}' \sim P(\boldsymbol{x}_t^i, a)$ determined. If $\boldsymbol{x}'$ is not already in $\mathcal{C}_a$, it is recalled from the hash table and added. This way, $\mathcal{C}_a$ will converge to the set of successors to all consistent samples. Leafs simply have no children, i.e. $\mathcal{C}_a = \varnothing, \forall a \in \mathcal{A}$. After the forward pass has reached a leaf, the backwards pass adjusts the value $V(\boldsymbol{x}_t)$ of all nodes along the sampled path:

$$V(\boldsymbol{x}_t) \quad := \quad \mathbb{E}\Big[\ln p_\theta(y^i \,|\, \boldsymbol{x}_t) \,\Big|\, i \in \mathcal{X}(\boldsymbol{x}_t) \Big] + \max_{a \in \mathcal{A}} \mathbb{E}\Big[V(\boldsymbol{x}') \,\Big|\, \boldsymbol{x}' \in \mathcal{C}_a \Big] \,. \tag{10}$$

Note that the average "reward" for classifying $\boldsymbol{x}_t$ does not depend on the choice of action a and can be computed once at the creation of the node. An obvious extension is to choose the executed action in the forward pass with some heuristic, e.g. with the learned π_ϕ.

5 Complexity Analysis

References

Gaudel, R. and Sebag, M. Feature selection as a one – player game. In *International Conference on Machine Learning*, pages 359–366, Haifa, Israel, 2010.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; Lillicrap, T., and Silver, D. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020. URL `https://arxiv.org/pdf/1911.08265.pdf`.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M., and Moritz, P. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, 2015. URL `http://proceedings.mlr.press/v37/schulman15.html`.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T., and Hassabis, D. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–503, January 2016. URL `http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html`.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T., and Hassabis, D. Mastering the game of go without human knowledge. *Nature*, 550:354–359, October 2017. URL `http://dx.doi.org/10.1038/nature24270`.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K., and Hassabis, D. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. URL `https://science.sciencemag.org/content/362/6419/1140`.

Sutton, R. S.; McAllester, D., and anf Yishay Mansour, S. S. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 12, pages 1057–1063. 2000.

Williams, R. J. Simple statistical gradient algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.