Mitigating The Failures Of Gradient-Based Program Synthesis

Jialin Lu

Abstract

This is still work in progress, as it seems that we need different tricks (different variations of the proposed algorithms) to solve different tasks, or at least different hyperparameter setting. It would be nice to find a good algorithm to solve all but I guess there is no free lunch after all. I think we might need to carefully rethink how to convert into a differentiable program in the first place. □

We revist the TerpreT [Gaunt et al., 2016] and in particular its parity chain task. We argue that the old approach of relaxing the discrete parameter into continuous space so to apply gradient descent is not satisfactory, as it is not guaranteed to converge to discrete values, which is hard to avoid, and also because conitnuous parameter configurations in the continuous space are not as informative as the discrete ones.

Here we argue an alternative approach is directly optimize in the discrete space. We derive Gradient-approximated Discrete Local Search (GradSearch), a generic gradient-based optimization procedure for discrete parameters. We use gradient to compute a Taylor series approximation of how much the objective function would decrease for a discrete local modification. GradSearch is **extendable**, we show that it can be easily extended from binary parameters to categorical variables. Some existing methods such as Bop [Helwegen et al., 2019] and Straight-through estimator [Bengio et al., 2013] can now be viewed as special cases of GradSearch. We also emphasize that GradSearch is **improvable** by injecting certain tricks or heuristics: This is because can be viewed as a mid point between the gradient-free discrete local search and gradient descent. This enable us to improve it by borrowing lessons and tricks from a wider literature of constraint search and numerical optimization.

However, the current state of GradSearch is only limited success: We show that by using a running average of gradient to replace the gradient as learning signal, we we can effectively solve the parity chain problem, while any continuous relaxation methods cannot optimize efficiently and will be stuck in soft local minima (the number of which grows expoentially with relaxation). We then show how we can work on some other challenging tasks such as learning the Boolean Circuits, disjunctive normal forms, though this time we need to incorporate other tricks. We suggest that in order to work on more general programs, we might need to more carefully invesigate how to handle the 'conditional' and 'loop', and perhaps need to modify how to translate a program into a differentiable one in the first place.

1 Introduction

We study the problem of gradient-based inductive program synthesis, i.e., optimizing the parameters of a program given input-output pairs, using gradient-based optimization algorithm. The task of inductive program synthesis can be seen as a type of machine learning where the hypothesis space is the set of possible programs. While conventionally tackled by discrete search-based algorithms, it is tempting to solve inductive program synthesis by gradient-based optimization because

Preprint. Under review.

1. Defining an objective function and then optimize is a simple and general workflow without using too many domain-specific knowledge and thus following the bitter lesson [Sutton, 2019], so hopefully we can potentially make it more scalable and faster.

2. It also opens the possiblities for hybrid models with continuous parameters and neurosymbolic programs [Chaudhuri, 2020].

Among machine learning approaches, gradient descent is the basic kind, so here we restrict ourself on gradient-based optimization. In a high level, our goal is to develop a method that can effeciently optimize various forms of discrete parameters, and ideally such a "discrete" optimizer can be pluged into a standard deep learning workflow to handle mixed discrete-conitnuous models and neurosymbolic models.

The most straight-forward and popular approach for it is the relaxation-then-gradient-descent approach. It first makes a differentiable version of the program, and then relaxes the discrete space of programs into a continuous space so that search over programs can be performed in the continuous space by gradient descent. This is usually done by representing a binary parameter by applying a sigmoid on a continuous parameter, or applying a softmax for a categorical variable.

However, against the wishful thinking of replicating the success of gradient descent for neural networks, the relaxation-then-gradient-descent approach for inductive program synthesis is neither effective nor scalable. Recent works [Feser et al., 2016, Gaunt et al., 2016] suggested that it is only capable of solving relatively simple problems and is outperformed by conventional approaches that are based on constraint solving and combinatorial search, often by a large margin. While it is not usually applicable to apply these more traditional methods on continuously parmaeterized and hybrid models, one should note that in order to pursue and develop mixed hybrid models, we must first show that gradient-based approach is able to achieve decent performance in discrete parameterized models.

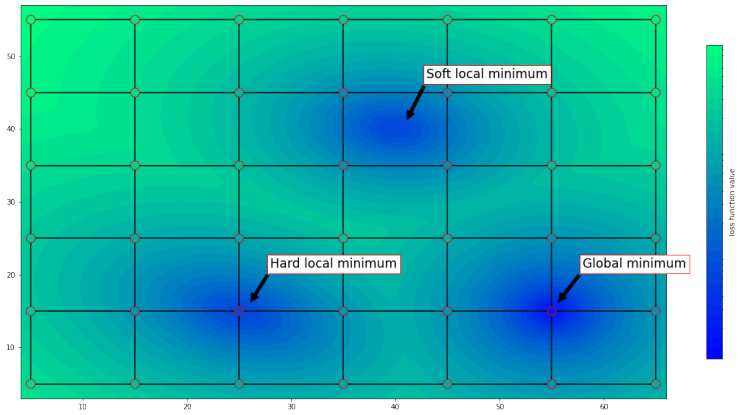

Figure 1: *Visualization of **hard** and **soft** local minima* Here we conceptually represent the discrete space of program as a grid of discrete points in the continuous space. When relaxing the discrete parameters into the continuous space, applying the gradient descent is vulnerable to both the *Hard* and *Soft* local minima. The latter leads to the poor results of parity chain problem. The relaxation-then-gradient-descent would spend most of its optimization trajectory in the continuous interior of the grids, instead of the more informative 'vertex' (discrete values).

The most striking failure is the parity chain problem from TerpreT [Gaunt et al., 2016]: while the alternative discrete search algorithms can solve very easily, but relaxation-then-gradient-descent gets stuck in soft local minima and thus fails constantly. It is found that the relaxation-then-gradient-descent approach can be stuck in *soft* local minima: the parameters do not converge to discrete values, but rather converge to a stationary point in the interior of a polytope. A straight-forward solution would be to enforce the parameters to have near-discrete values. However, some ad-hoc solutions such as regularizations or temperatured-softmax are demonstrated o be of litte to no help (see table 2).

Here we argue an alternative approach for gradient-based program synthesis: we still convert the to-be-inferred program into a differentiable version, but we do not further relax the parameter into continuous space so to apply gradient descent. Instead we keep the discrete restriction and perform optimization directly in the discrete space. The motivation for diectly search in the discrete space instead of continuous space is two-fold

- **the problem of soft local minima is hard, if not impossible, to avoid if we optimize in the continuous space**, because the program sketch can be highly flexible and arbitrary. It will be hard to rule out all programs that can cause soft local minima in the design phase of program specification (the sketch).
- **conitnuous parameter configurations in the continuous space are not be so faithful to the intended computation**, and error may accumulate because of the continuous relaxation. Thus optimization in the continuous space may not be efficient as expected. There is to date no golden standard on how to translate an ordinary program into a differentiable one, and usually the functions used are only well-behaving and tested on the vertex of the polytope (the discrete values). However, the conitnuous optimization will spend most of its optiimzation trajectory in the interior of the parameter polytope, but not in the vertex of the polytope. And the interior of the polytope would not be as informative as the vertex of the polytope. By optimizing directly in the discrete space, we should be able to search more effeciently. One similar but not-so-close analogy is the use of simplex algorithm in linear programming, though in linear programming the analysis of algorithm should be much easier than an arbitrarily specified differentiable program in our case.

We derive Gradient-approximated Discrete Local Search (GradSearch), a gradient-based optimization procedure for discrete parameters. The idea is that although the parameters are restricted to discrete values, the gradient are well defined in the continuous space for a differentiable program. So we can use gradient at the discrete points to compute a Taylor series approximation of how much the objective function would decrease for a discrete local modification. GradSearch is

1. **Extendable**: It can be easily extended from binary parameters to categorical variables. Some existing methods such as Bop [Helwegen et al., 2019] and Straight-through estimator [Bengio et al., 2013], which are very sensitive to hyperparameter and confined to binary parameters, can now be viewed as special cases of GradSearch.
2. **Hackable** or **improvable**: GradSearch can be viewed as a mid point between the gradient-free discrete local search and gradient descent. This enable us to improve it by borrowing lessons and tricks from a wider literature of constraint search and numerical optimization.

We are aware that by keeping the discreteness of the parameters, GradSearch comes at a price as it loses continuity and smoothness, and become difficult to model and analyse, for example, convergence. So it shoule be no surprise that the optimization will encounter many problems and become very unfriendly. However, by inspiration and lesssons from a wider literature of constraint search and numerical optimization, we show that it is possible to mitigate the failure of gradient-based program synthesis.

In section 2 we introduce the formulation of GradSearch . In section 3.1 we show how we can use the running average of gradient to effectively solve the parity chain problem. In section 3.2 we start to solve the challenging Boolean Circuits tasks, but with limited success. In section 3.3 we provide some other experiemnts.

2 Gradient-approximated Discrete Local Search (GradSearch)

2.1 What can be ideal

Our goal is to develop a method that can effeciently optimize various forms of discrete parameters, and ideally such a "discrete" optimizer can be pluged into a standard deep learning workflow to handle mixed discrete-conitnuous models and neuro-symbolic models. Ideally, we should not use any domain-specific knowledge on the structure of the program except the gradient information, and the algorithm should be robust to hyperparameters.

2.2 The proposed algorithm

Here we derive our GradSearch algorithm. At a certain iteration, we consider all local modification of hamming distance 1 (so for binary variables, it is equal to flip one dimension)

The notable difference of GradSearch algorithm algorithm 1 and a standard discrete local search algorithm is in line 3, where we use the taylor series approximate $(v(\theta_i) - \theta_i)^T \vec{\nabla}_{\theta_i}$ to estimate $\mathbb{L}|_{\theta_i}^{v(\theta_i)} = \mathbb{L}(v(\theta_i)) - \mathbb{L}(\theta_i)$. In standard discrete local search, the loss function is evaluated for each possible local modification, while in our GradSearch , we only evaluate the loss function once and use a Taylor approximation to estimatet it. As the gradients can be efficiently computed by automatic differentiation, this approximation cost only a small computation.

Algorithm 1 A generic Gradient-approximated discrete local search procedure (but does not work so well). The neighbourhood is defined to the Hamming distance of 1. Given a differentiable program, we can approximate the potential decrease of loss function with gradient, and it requires evaluating the loss at θ_i only once with the help of automatic differentiation.

Input: loss function $\mathbb{L}$; initial parameter θ_0

1: **for** $i = 0, 1, 2, \ldots$ **do**
2: **for** every $v \in V(\theta_i)$ **do**
3: compute $\mathbb{L}|_{\theta_i}^{v(\theta_i)} \approx (v(\theta_i) - \theta_i)^T \vec{\nabla}_{\theta_i} \mathbb{L}$
4: Update. Let $\theta_{i+1} = v(\theta_i)$ where $v = \operatorname{argmin}_v \mathbb{L}|_{\theta_i}^{v(\theta_i)}$.

Figure 2: *The Gradient-approximated version of ours.* When the program is made differentiable and with the help of automatic differention, we can approximate the potential decrease of loss function with only one evaluation of the loss function $\mathbb{L}$.

2.3 Rules for approximation (line 3)

Here we give the approximation rule for binary variables and categorical variables.

For a binary variable $w \in \theta$, $\theta_{\text{flip } w}$ represent the parameter when the binary variable w is flipped from 0 to 1 or 1 to 0.

$$\mathbb{L}|_{\theta}^{\theta_{\text{flip } w}} \approx (1 - 2w)\nabla_\theta \mathbb{L}$$

For a M dimensional categorical variable $w_{M\times 1}$ (w represnted as a one-hot coding column vector), $\theta_{\text{flip } w,j}$ represent the parameter when the categorical variable w is modified to its j-th dimension being one.

$$\mathbb{L}|_{\theta}^{\theta_{\text{flip } w,j}} \approx \nabla_\theta \mathbb{L}_j - w^T \nabla_\theta \mathbb{L}$$

The use of taylor approximation to guide local search is a simple idea, and putting it to actual use is not without problem: the taylor series approximation is an approximation after all. To motivate the use of this taylor approximation, we might either show that this approximation works well in practice, or that this is effcient in the sense we can afford more steps of update given its relatively cheap computation. But in our case, since we are ultimately interested in finding a gradient-based optimization algorithm for discrete parameters and ideally would like to use it along with another conitnuous optimizer to joinly optimize a hybrid conitnuous-discrete model or neurosymbolic programs, this might not a bad idea as it looks. More overall, we might just use tricks from numerical optiimzation to improve it, as is the case in section 3.1 where we use the running average of the gradient in order to effectively solve the parity chain problem. This being said, we are not aware of

the application of it in the case of inductive program synthesis. We do know that this trick has been applied to SAT solvers. G^2WSAT [Li and Huang, 2005] is one of the most representative SAT solver in the WalkSAT family. It uses gradient-based greedy heuristic guide the search by first formulating a cost function and taking gradient with taylor approximation (which is in turn originally introduced in Huang and Chao [1997]). It is efficient to compute and can further improve the performance in the WalkSAT family.

2.4 heuristics and tricks

Search methods are often made with heuristics as real world applications can rarely be tackled without any heuristics. algorithm 1 is also just a generic algorithm without any heuristics, and we expected that we empoly various tricks in order to make it work.

However, as for now we have not find a once-for-all heuristic to solve every task. As we will show in the experiments, we show that by using a running average of gradient to replace the gradient as learning signal, we we can effectively solve the parity chain problem. As for the Boolean Circuits, we find that we need to use some other tricks to achieve limited success.

2.5 Connection to Bop and STE

We can show that Bop [Helwegen et al., 2019] and STE [Bengio et al., 2013] are special cases of our algorihthm for binary parameters of $\{0, 1\}$ or $\{-1, 1\}$. The most important insight is that, in our algorithm algorithm 1, we constrain the considered local modifications to be of hamming distance 1. If all parameters are binary parameters, as there is only one possible filp for each binary parameter, each local modifications then can be aggregated together to update at once. So we can replace the update rule in algorithm 1 from only choosing the most greedy local move $v = \underset{v}{\operatorname{argmin}} \mathbb{L}|_{\theta_i}^{v(\theta_i)}$, to consider all possible move v as long as the local move meets some criteria, such as a threshold:

$$\gamma > -(v(\theta_i) - \theta_i)^T \vec{\nabla}_{\theta_i} \mathbb{L}$$

where $\gamma \geq 0$ is a hyperparameter representing a threshold for accepting. So if $\gamma = 0$, we will accept any update as long as the taylor approximated decrease is smaller than zero. This then corresponds to the well known straight-thorough estimator; where a $\gamma > 0$ will correspond to the Binary optimizer [Helwegen et al., 2019].

Algorithm 2 Bop/STE

Hyperparam: $\gamma \in [0, 1]$ - the threshold parameter; **Input**: loss function $\mathbb{L}$; initial parameter θ_0

for $i = 0, 1, 2, \ldots$ **do**
  for every $v \in V(\theta_i)$ **do**
    compute $\mathbb{L}|_{\theta_i}^{v(\theta_i)} \approx (v(\theta_i) - \theta_i)^T \vec{\nabla}_{\theta_i} \mathbb{L}$
  for every $v \in V(\theta_i)$ **do**
    Accept v as long as where $\gamma > -(v(\theta_i) - \theta_i)^T \vec{\nabla}_{\theta_i} \mathbb{L}$.

Figure 3: *Bop/STE* This update rule is possible for binary variables because each move v is not conflicted with each other, as each v is a flip of a bit, so it is possible to apply multiple local move v at once. STE is a special case of Bop where $\gamma = 0$

3 Experiments

We find using mean squared error or binary cross entropy does not make much difference. Through out the experiment we use binary cross entropy as the loss function.

3.1 Parity Chain

We refer to the original TerpreT [Gaunt et al., 2016] and this blog [Selsam, 2018] as a quick reading for what the parity chain task is and how it is difficult to solve. The parity chain has a problem size k which denotes the length of the parity chain, the larger k is, the more difficult it is. In short, the continuous relaxation approach happens to be stuck in soft local minima where the parameters

Table 1: *Percentage of success and average steps of convergence among success runs on parity chain*
♣ represents the methods that are parameterized or reparameterized by continuous parameters.
♠ represents the methods that do not modify the forward computation.
★ are the methods are directly optimized in the discrete space.

	K=4	K=8	K=16	K=32	K=64	K=128	K=256
continuous relaxation♣ (FMGD [Gaunt et al., 2016])	100%/1077.72	65%/1278.51	12%/1260.00	0%/NA	0%/NA	0%/NA	0%/NA
continuous relaxation Regularized♣	100%/785.13	61%/1157.39	16%/1206.00	0%/NA	0%/NA	0%/NA	0%/NA
Temperatured softmax♣	100%/343.96	61%/376.44	13%362.08	0%/NA	0%/NA	0%/NA	0%/NA
Temperatured softmax regularized♣	97%/341.52	66%/365.83	27%/378.78	0%/NA	0%/NA	0%/NA	0%/NA
Gumbel-softmax♣	100%/556.00	87%/686.34	27%/805.22	0%/NA	0%/NA	0%/NA	0%/NA
Neural reparameterization♣	100%/350.18	70%/193.07	22%/192.05	3%/91.00	0%/NA	0%/NA	0%/NA
STE♣♠	100%/49.36	100%/82.56	100%/207.64	88%/996.51	29%/1167.62	3%/1999.00	0%/NA
Bop★♠	100%/11.97	100%/15.55	100%/22.23	100%/35.89	100%/63.14	100%/116.96	100%/223.58
GradSearch ★♠	100%/11.86	100%/19.34	100%/88.76	100%/271.47	88%/743.52	21%/963.57	9%/2618
GradSearch (running average)★♠	100%/11.56	100%/16.78	100%/36.07	100%/74.93	100%/178.16	100%/399.88	100%/1076.82

forms isolated islands of 0's and 1's which are seperated by a soft value of 0.5. It just happen to be a stationary point where the gradient converges to zero.

The straight-forward solution to *type 2* local minima is to enforce some regularizations to encourage the parameters to have near-discrete values. What is more intriguing, it happens to be the case of the parity chain problem that the continuous relaxation will get stuck in somewhere in the middle value being 0.5. but ironically such regularization is usually symmetrical, meaning Without any prior information, regularization is usually symmetrical, meaning it encouranges equally for a binary ($\{0, 1\}$) variable to be close to 1 or 0. and thus cannot mitigate this situation. We test a wider range of possible approaches such as regularization and reparamaterizations, but unfortunately these approaches do not solve the parity chain task.

We then also test other approches such as STE and Bop which do not modify the forward computation by continuous relaxation, but use discrete values. We see that while STE does not provide a decent performance, we do can make Bop solve the parity chain with a carefully tuned hyperparameter γ. Though I would think this success is rather a abnormaly, as it is highly sensitive to the hyperparameter configuration: if we change the value of γ it just would not work. What is more, Bop is confined to binary variables.

As for the GradSearch , the vanilla version does can solve parity chain under a small problem size It is found out that STE and Bop are in fact quite efficient empirically for binary neural networks. And using a running average of gradient is also very important. Here we are going to use the running average of gradient as well, as it can be seen as a priliminary way to reducing variance.

$$g_{v,t} = (1-\beta)g_{v,t-1} + \beta(v(\theta_i) - \theta_i)^T \vec{\nabla}_{\theta_i} \mathbb{L}$$

And so the update rule is still $v = \underset{v}{\operatorname{argmin}}\, g_{v,t}$. We can find that we can effectively solve the parity chain task. Although looking at the visualization of the optimization trajectory should be more interesting as we can find some propogation pattern (not shown here).

3.2 Boolean Circuits

The Boolean circuits are a set of tasks of different difficulty level from the TerpreT. It is shown that the boolean circuit tasks are quite difficult for the relaxation-then-gradient-descent approach, as none of the circuit tasks can be solved by gradient descent.

However, not like the parity chain, we find that using the running average does not magically solve the circuits. But the good news is that, as we stated earlier, we are able to borrow certain tricks from a wider range of literature to make it work. So by incorporating the following tricks (some painful trial-and-error), we are able to solve the first Boolean Circuit tasks, and the success rate and synthesis speed is comparable to the state-of-the-art program synthesizer Z3 and Sketch.

Denote this improved version as GradSearch $^+$, we identify the following problems which GradSearch can face and uses some tricks and heuristics to solve the corresponding problem

- **Problem: Local minima**, any local search faces this problem. There are many ways to mitigate this problem, but I fear that the multiple-restart strategy is not so deep learning, so we employ a ϵ-greedy heuristic, with probability $P = 0.1$, we choose to randomly apply a local modification instead of the greedy one.
- **Problem: Zero gradient**, in particular, the conditional statement is expressed by a weighted sum in the TerpreT paper, but when we use discrete parameters, we are effectively setting all but one condition to be zero, thus the larger portion of the circuit below will not receive any gradien. We mitigate this by applying a (rather very suspicious trick) smoothing by treating all 1's as 0.99 and all 0's as 0.01. So if a two-way conditional experssion is parameterized by $[1, 0]$ we replace it by $[0.99, 0.01]$.
- **Problem: hard to give a name**. During out experiment, we find that in certain cases if we have multiple data samples and multiple output dimensions for each data sample, then it is possible that the synthesized program can solve a subset of the data samples or output dimensions, so the optimization will be entering a stage of stalemate. In this case, we find we can exponentially scale the loss function to certain data samples randomly to partially mitigate the problem. This trick can be understood as a variant of dynamic local search by adjusting the loss function. I think perhaps this is probably a great space of opportunities.

By applying all these tricks, we find we can solve the first two circuits tasks with 100% of success rate and a decently fast speed. However, for the latter three challenges tasks, we are still not able to solve.

Table 2: *Benchmark results on Boolean circuit tasks* The evaluation should be whether success (or percentage of success) for 100 runs and the wall time to reach a solution. The time is the average wall time for the success runs, we do not include the failed runs in the time. '$\times$' reprents cannot be solved for one hour.

	continuous relaxation	GradSearch	GradSearch $^+$	.....	SMT Z3	Sketch
Circutis NAND	$\times$	0.12s/35%	0.43s/100%		0.138	1.40
Circutis Swap	$\times$	$\times$	2.05s/100%		0.10	1.36
Circutis Shift	$\times$				15.57	1.90
Circutis Adder	$\times$				17.719	2.55
Circutis Full Adder	$\times$				$\times$	$\times$

We can find that the vanilla version of GradSearch has only a success rate pf 35%, and the run time of successful runs is suspiciously small. This suggests nothing but that the successful runs might just happened to have a good initialization. The trick-enhanced GradSearch $^+$ however, is able to solve the first two circuit reliably. (Although I am aware all these tricks combined is just not very good looking).

3.3 Other tasks.

We also managed to optimize a disjunctive normal form by using the trick-enhanced version of GradSearch as in last section.

We also managed to optimize a conditional SVG program, by defining three shapes and a conditional command to choose one of the shapes. This is a proof-of-concept experiment to show that we can jointly optimize a hybrid model consisting of a categorical parameter along with some other continuous parameters.

There are also many other experiments we would like to conduct, such as using an arbitrary loss function such as regularization on the complexity of a program, program synthesis under a noisy setting, and ultimately testing a program-network hybird model.

References

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Swarat Chaudhuri. Synthesizing Neurosymbolic Programs. `https://blog.sigplan.org/2020/04/15/synthesizing-neurosymbolic-programs/`, 2020.

John K Feser, Marc Brockschmidt, Alexander L Gaunt, and Daniel Tarlow. Differentiable functional program interpreters. *arXiv preprint arXiv:1611.01988*, 2016.

Alexander L Gaunt, Marc Brockschmidt, Rishabh Singh, Nate Kushman, Pushmeet Kohli, Jonathan Taylor, and Daniel Tarlow. Terpret: A probabilistic programming language for program induction. *arXiv preprint arXiv:1608.04428*, 2016.

Koen Helwegen, James Widdicombe, Lukas Geiger, Zechun Liu, Kwang-Ting Cheng, and Roeland Nusselder. Latent weights do not exist: Rethinking binarized neural network optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 7533–7544. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/9ca8c9b0996bbf05ae7753d34667a6fd-Paper.pdf`.

Wen Qi Huang and Jin Ren Chao. Solar: A learning from human algorithm for solving sat. *Science in China (Series E)*, 27(2):179–186, 1997.

Chu Min Li and Wen Qi Huang. Diversification and determinism in local search for satisfiability. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 158–172. Springer, 2005.

Daniel Selsam. The "Terpret Problem" and the limits of SGD. `https://dselsam.github.io/the-terpret-problem/`, 2018.

Richard Sutton. The bitter lesson. *Incomplete Ideas (blog), March*, 13:12, 2019.

A implementation

It is important that the argmax is implemented with a random tie-breaking. Otherwise the greedy version of GradSearch will get stuck in cancel and fail constantly. Using a random tie-breaking for greedy gradsearch does not solve the problem, as it is still stuck in a local basin of attraction.