Online Learning for Faster Verification of Neural Nets

Speeding up integer programs with online convex optimisation

Lewis Smith (lsgs@robots.ox.ac.uk), M. Pawan Kumar (pawan@robots.ox.ac.uk)

UNIVERSITY OF OXFORD

Neural Network Safety

- Deep nets are our most empirically successful models in a lot of applications by a large margin
- But **not robust**: adversarial examples are easy to find. Small changes to an input can have a large effect on the model response
- Leads to interest in *provable verification* of such models

Verifying Neural Nets

- Neural net is a parameterised function; $x_n = f(x_0)$
- Consider properties that are linear inequalities on the outputs, for the input lying in some domain $\mathcal{C}$;

$$P := x \in \mathcal{C}, x_n = f(x_0) \Rightarrow c^\mathsf{T} x_n - d \geq 0$$

- Can prove or disprove this by solving an optimisation problem

$$\begin{aligned} \text{min} \quad & c^\mathsf{T} x_n - d \\ \text{s.t} \quad & x_n = f(x_0) \\ & x_0 \in \mathcal{C} \end{aligned}$$

- If the value of this program is above zero, then the property holds. If not, we have found a counter-example.
- **Problem**: this optimisation problem is non-convex, and the above is only true if we can find the *global* optimum. In fact, turns out to be NP hard!

Branch and Bound

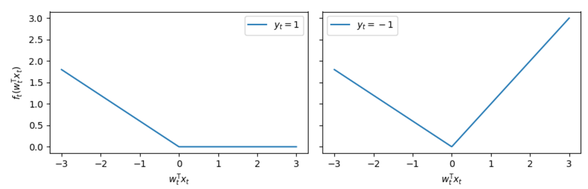
Input: Objective function f, domain $\mathcal{C}$, tolerance ϵ
$ub \leftarrow$ compute_ub($\mathcal{C}$)
$lb \leftarrow$ compute_lb($\mathcal{C}$)
doms $\leftarrow \{(lb, \mathcal{C})\}$
while $ub - lb > \epsilon$ **do**
  (_,dom) $\leftarrow$ pick_out(doms)
  $[d_1, d_2 ... d_n] \leftarrow$ split(dom)
  for $i = [1..n]$ **do**
    $ub' \leftarrow$ compute_ub(d_i)
    $lb' \leftarrow$ compute_lb(d_i)
    if $ub' < ub$ **then**
      $ub \leftarrow ub'$
      doms $\leftarrow \{(l, d) \mid (l, d) \in \text{doms}, l < ub\}$
    end if
    if $lb' < ub$ **then**
      doms $\leftarrow (lb', d_i) \cup$ doms
    end if
  end for
  $lb \leftarrow \min\{l \mid (l, d) \in \text{doms}\}$
end while

- General strategy for combinatorial problems.
- Basic idea; split our original domain into a series of sub-domains by introducing extra constraints.
- Can easily compute *upper bounds* by choosing any feasible x_0 and evaluating the objective
- Can compute *lower bounds* by solving a linear program, formed by relaxing the non-linearities of the network.
- Use lower and upper bounds to try to rule out the possibility that a sub-domain contains the global minimum
- In the worst case, still have to explore all the states, but if we have good approximations and heuristics can often be sufficiently fast

Online Budgeting

- From previous work; use a cheaper, but looser lower bound, calculated by choosing a feasible point in the dual (without optimisation) to decide which dimension to split the input on
- **Problem**: Bottleneck of this procedure is that we still have to solve a very large number of linear programs.
- But do we actually need to optimise the lower bound every time?
- **Idea**: predict whether we should compute the expensive, tighter lower bound for a branch at every iteration.
- Basically, it's worth computing the tighter bound if it pushes the lower bound for that region above zero, and we can prune out that domain. Otherwise, the computation is wasted, as we will continue splitting on the domain anyway.
- Try to learn this *online*, adapting the model for the specific problem we are trying to solve.
- The problem is in the training data; we only know the ground truth if we choose to evaluate the LP
- Solution; use a simple, linear decision sign$(w_t^\mathsf{T} x_t)$ and, defining a variable y_t = 1 if the true LP bound is above zero and $y_t = -1$ otherwise, use the loss

$$f_t(w_t) = \begin{cases} \max(-\alpha w_t^\mathsf{T} x_t, 0)) & y_t = 1 \\ \max(-\alpha w_t^\mathsf{T} x_t, \beta w_t^\mathsf{T} x_t) & \text{otherwise} \end{cases}$$



- Acts like the hinge loss when y_t is positive, and we are getting the ground truth anyway.
- When y_t is negative, this cost is minimised when the decision function is exactly 0 on that example; penalises budgeting confidently, and but also punishes evaluation of negative points.
- Key feature; in the region where $w_t^\mathsf{T} x_t < 0$, the decision function does not depend on the sign of y_t. This allows applying it even though we don't know y_t in this situation.

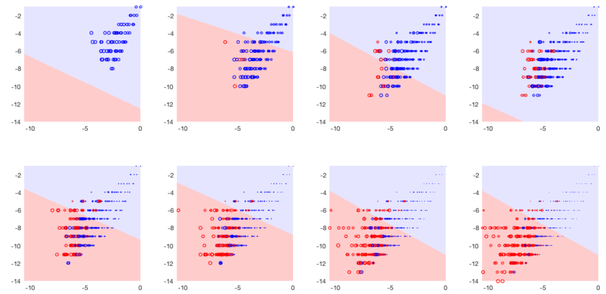


Figure 1: The online algorithm in action on data where the ground truth y_t values were computed by brute force. Red corresponds to $y_t = 1$. The size of points is proportional to how recently they were observed; more recent points are larger. We see that the algorithm mostly maintains a reasonable decision boundary. The features are the (scaled) cheap lower bound on the x axis and the area of the domain on the y. Both quantities are used in a log scale.

Results

		STD (s)	B1 (s)	REL	NWins	B2 (s)	REL	NWins
ACAS	ALL	59697	28173	47%	175/188	24267	41%	168/188
	SAT	33370	6269	19%	36/37	1442	4%	36/37
	UNSAT	26327	21904	83%	139/151	22825	87%	132/151
PCAMNIST	ALL	1440	972	68%	12/16	1160	81%	12/16
	SAT	14	6	43%	4/4	10	71%	4/4
	UNSAT	1426	966	68%	8/12	1150	81%	8/12

Online Convex Optimisation

- Online convex optimisation; playing a repeated game against an adversary. Possible actions form a convex set $\mathcal{W}$, cost functions are convex and belong to a bounded family $\mathcal{F}$.
- At each round t, choose an action $\omega_t \in \mathcal{W}$, and observe a cost $f_t \in \mathcal{F}$.
- Sequence of cost functions is arbitrary, and can even be adversarial chosen; imagine we are playing against a demon who knows our strategy and is able to choose a sequence to make our lives as hard as possible.
- Aim to minimise the *regret*; the worst case difference between our choices, as determined by algorithm $\mathcal{A}$, and the best fixed action chosen with hindsight;

$$\text{regret}_T(\mathcal{A}) = \sup_{\{f_{1..T}\} \in \mathcal{F}} \left[\sum_{t=1}^{T} f_t(\omega_t) - \min_{\omega \in \mathcal{W}} \sum_{t=1}^{T} f_t(\omega) \right]$$

- An online algorithm is useful if it has *sublinear regret*: $\text{regret}_T(\mathcal{A}) = o(T)$, which implies that our average regret converges to the best fixed strategy.