Exploring Boosted Recurrent Neural Nets For Rubik's Cube Solving

Alex Irpan

UC Berkeley, Computer Science

Objectives

We explore integrating AdaBoost with neural net training. AdaBoost focuses training on difficult examples, which could train faster than uniform weighting. It also gives a natural ensemble for further boosting performance. We examine whether AdaBoost achieves these results, using the problem domain of solving a Rubik's Cube.

Background

Exhaustive search has shown every cube is solvable in at most 26 turns. The best search algorithm is Kociemba's Two Phase Algorithm, which uses iteratively deepened A* search guided by domain knowledge [3].

Problem Setup

Model the Rubik's Cube as an MDP, and treat finding an solution as a sequence to sequence problem. An episode is K random moves, and we want the net to output the inverse of those moves. Changing K tunes difficulty. Input is a one-hot encoding of sticker colors. 54 stickers and 6 colors gives 324 inputs.

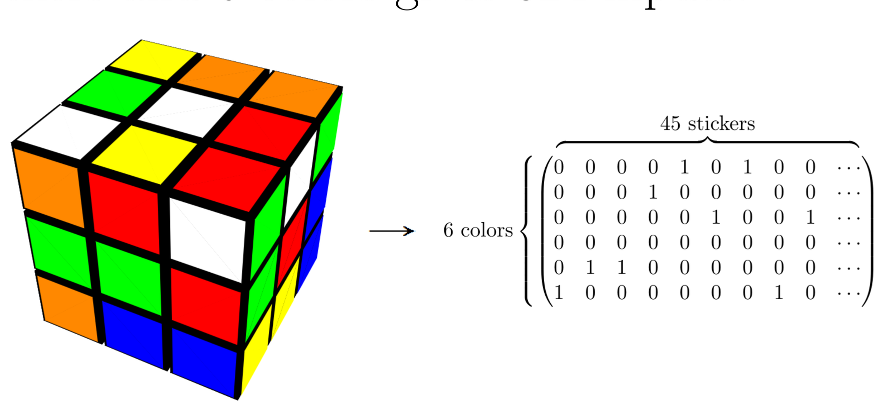

Figure 1: Sample One-Hot Encoding of Cube State

Difficulties

The Rubik's Cube domain has some interesting properites.

- We have access to arbitrary amounts of data, and it is easy to tune difficulty.
- However, episode labels are not 100% accurate. A given cube has many possible solutions.

In the *filtering* setting, algorithms assume access to a sampling oracle. We combine aspects of filtering boosters and AdaBoost to get a custom boosting algorithm that fits our use case.

Algorithm

We modify AdaBoost.M2 to sample new data every timestep. Weak learners must output a confidence vector over classes. $f : (X, Y) \to [0, 1]$.

AdaBoost.M2 turns (x_i, y_i) into mislabelings $\{((x_i, y_i), y)\}$ for every wrong y. AdaBoost then expects weak learner f to discriminate y_i from y.

$$\text{ploss}(f, (x_i, y_i), y) = (1/2)(1 - f(x_i, y_i) - f(x_i, y))$$

For any distribution D over mislabelings, require an edge over random guessing. ($\mathbb{E}_D[\text{ploss}(f, \cdot, \cdot)] < \frac{1}{2}$)

Algorithm 1 Refreshed AdaBoost

Input: Dataset size n, oracle Sample(), $p \in [0, 1]$

Init $\{(x_i, y_i)\}$ with n calls to Sample()
For $i = 1, \ldots, n, y \in Y \setminus \{y_i\}$, $D_1(i, y) = \frac{1}{|Y|-1}$
for $t = 1$ to T **do**
  Train weak learner f_t with distribution $D_t(i, y)$
  $\epsilon_t \leftarrow \frac{1}{n} \sum_{(i,y)} D_t(i, y)\, \text{ploss}(f_t, (x_i, y_i), y)$
  $\alpha_t \leftarrow \log \frac{1-\epsilon_t}{\epsilon_t}$
  $F_t \leftarrow \frac{1}{Z_t} \sum_{s=1}^{t} \alpha_s f_s$, where Z_t is a normalization constant
  for $i = 1$ to n **do**
    $D_{t+1}(i, y) \propto D_t(i, y) e^{\alpha_t(1+\text{ploss}(f_t,(x_i,y_i),y))}$
  Normalize to make $\sum_{(i,y)} D_{t+1}(i, y) = n$
  for $i = 1$ to n **do**
    Keep sample (x_i, y_i) with probability p.
  $m \leftarrow$ num samples lost, $W \leftarrow$ weight lost
  Call Sample() m times to get new (x_j, y_j)
  $D_{t+1}(j, y) \propto e^{\sum_{s=1}^{t} \alpha_s \text{ploss}(F_t,(x_j,y_j),y)}$
  Normalize to make $\sum_{(j,y)} D_{t+1}(j, y) = W$
return F_T

Convergence Theorem

$$\mathbb{E}[0/1 \text{ loss of } F_T] \le (|Y| - 1) \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1 - \epsilon_t)}$$

Prove by reduction to two class AdaBoost [2]. Still holds in expectation because the replaced data is sampled from the same distribution.

Neural Net Training And Boosting

The weak learner is a neural net. We train a single neural net over all t, multiplying gradients by sample weight. Save snapshot of net for each t. For efficiency, we only keep the models with largest α_t.

Architecture Choice

Initial experiments showed LSTMs outperforming RNNs and fully connected nets, even when all had the same number of parameters. Classification accuracy directly led to improved solving ability.

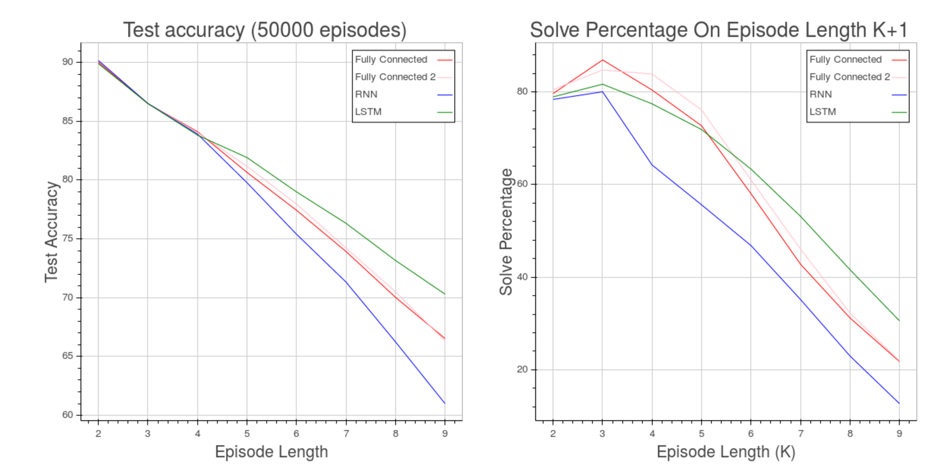

Figure 2: Classification Acc. and Solve % of Varying Architectures

The final architecture uses an LSTM of 100 hidden units. Used $n = 50000$ and episode length $K = 9$.

Results

Net trained with AdaBoost guidance did worse than baseline, and took longer to train.

Method	Accuracy	Run Time
Baseline	72.26%	237 min
Boosted, $p = 0.8$	67.56%	366 min
Boosted, $p = 0$	67.91%	439 min

Table 1: Experiment Results

Baseline also has better solve percentages across the board. Some generalization, but solve rate drops quickly.

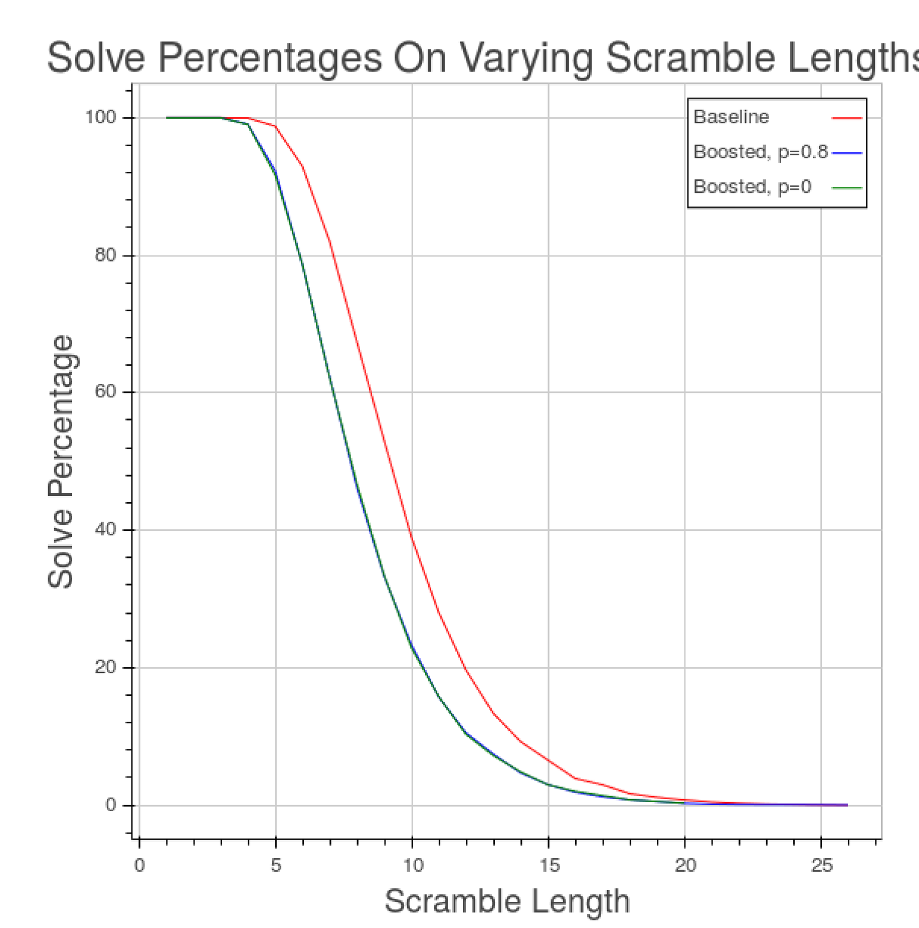

Figure 3: Solve Percentages of Best Model From Each Method

Analysis

Intuitively, AdaBoost should work better when

- Weak learners are cheap to train and evaluate.
- Weak learners propose different outputs when given the same input

Neural nets are not cheap to train, and working with the same neural net means the net is regularized against changing too much each iteration. Manual inspection showed neural nets tended to agree on best label.

Theoretically, approach will work with enough iterations, but it is too computationally expensive and there are not enough gains.

Current approach does not deal with fuzziness in episode labels well. Further work needs to address that multiple moves are valid.

Extensions

- Use reinforcement learning to optimize solve percentage. (RL better suited to discrete reward.)
- Curriculum learning - slowly increase K as net acheieves performance.
- Adjust loss and labels if neural net predicts a valid solution that differs from ours.

References

[1] Bradley, Joseph K. "FilterBoost: Regression and Classification on Large Datasets." 2007.

[2] Freund, Yoav, and Robert E. Schapire. "A desicion-theoretic generalization of on-line learning and an application to boosting." 1995.

[3] Kociemba, Herbert. "Two-Phase Algorithm Details." 2014. Web. 27 Apr. 2016.

More Information

- Code: github.com/alexirpan/rubik_research
- Email: alexirpan@berkeley.edu