Blackbox challenge

5 vision solution

Pavlov Mikhail
5vision team

11 June 2016

Plan

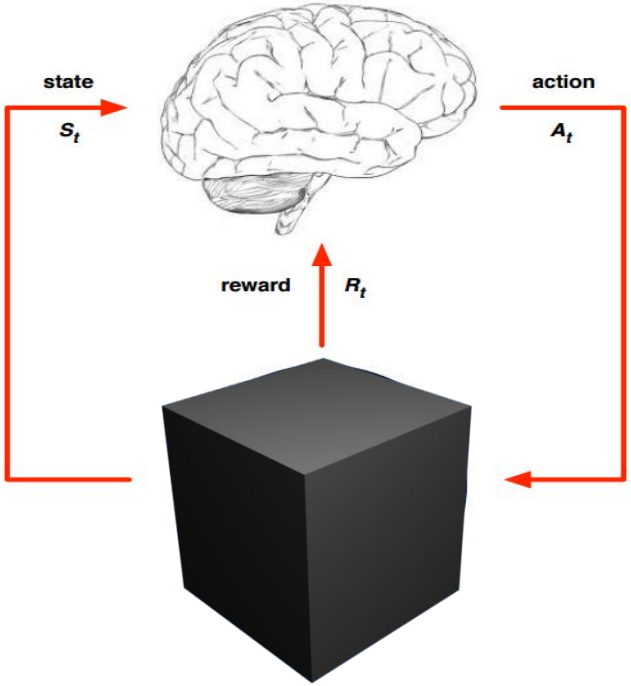
Part A: Black box problem

Part B: 5vision solution

Part C: Overview of other participants solutions

Part A: Black box problem

A. Black box problem. General Info



Website: http://blackboxchallenge.com/home/

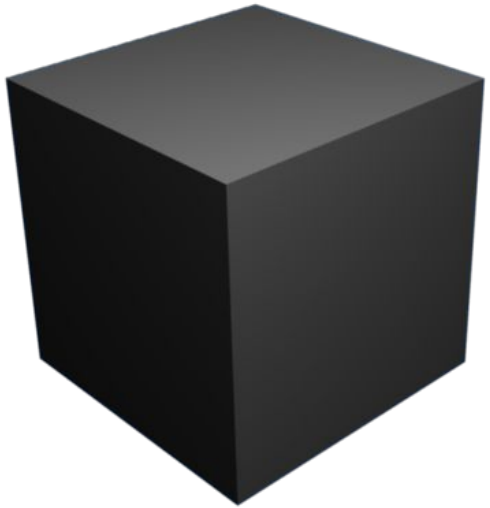
Problem description: The BlackBox Challenge is an open machine-learning and AI programming competition in which competitors teach an agent to play a game having unknown rules.

Dates: 1 March - 30 May

Game mechanics:
Each timestep agent:
- receives blackbox state S_t (dim 36)
- receives reward signal R_t(scalar)
- choose one action (dim 4)

agent objective - choose actions that lead to maximization cumulative reward signal - sum(R_t)
number of timesteps for level~ 1 200 000

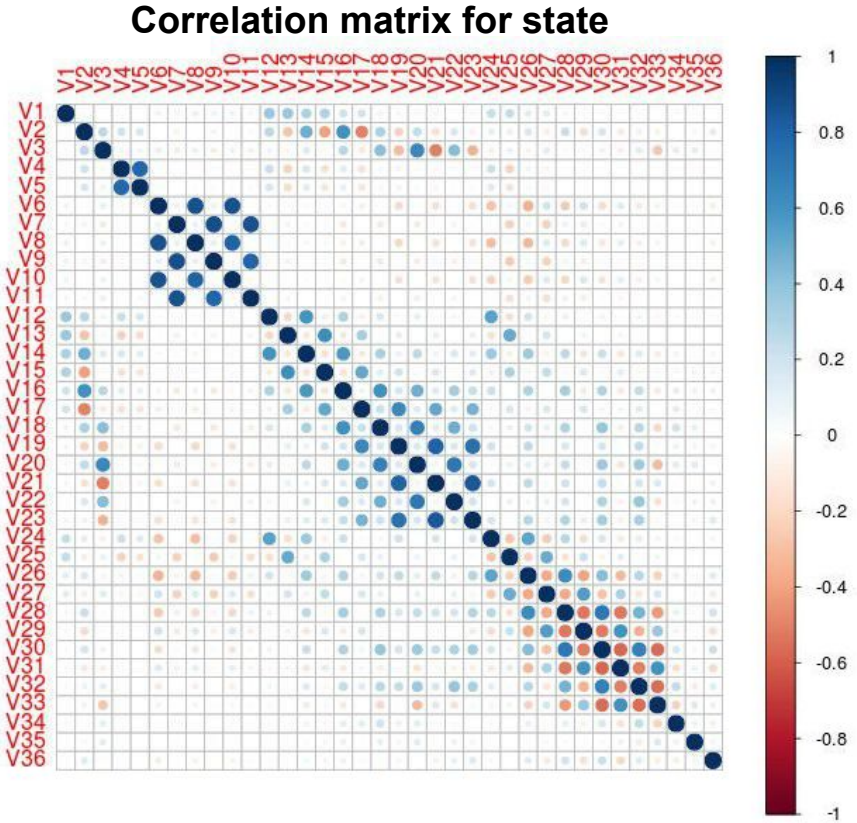
A. Black box problem. General Info



Data:
4 levels:
tran(given), test(given),
validation(leaderboard), final(final evaluation)
all levels follow the same mechanics

Baseline agent/solution:
organizers provided baseline agent, without details how to train it to encourage interesting solution.
Only ~100 participants out of 1380 were able to beat it

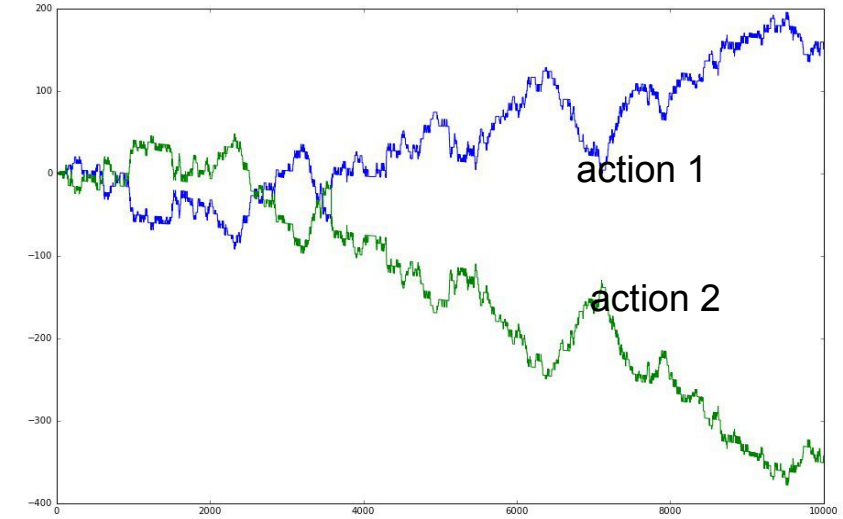
A. Black box problem. State and actions

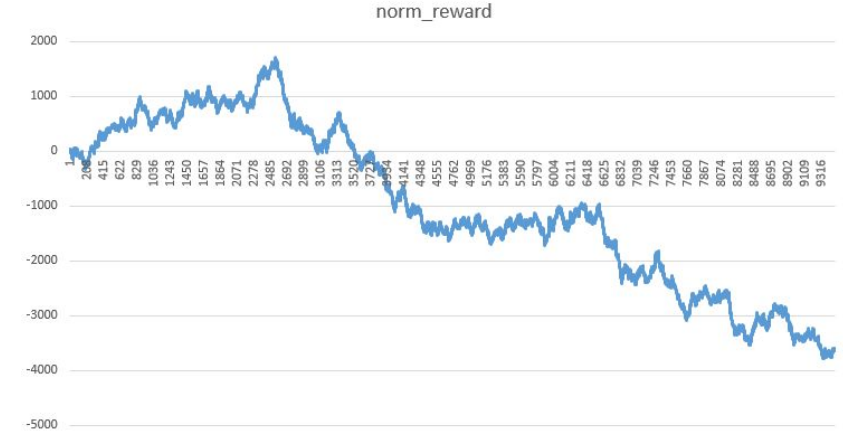


Blackbox state and actions:

- first 35 components of state are fixed for each level and each time step.
- actions affect only 36th component (state[35]), with low probability ~ 95% (possibly actions have some effect on hidden variables)
- some state components are correlated
- action 0 - increase/decrease based on values of other components
- action 1 - increase state[35]
- action 2 - decrease state[35]
- action 3 - no effect on state[35]

A. Black box problem. Reward



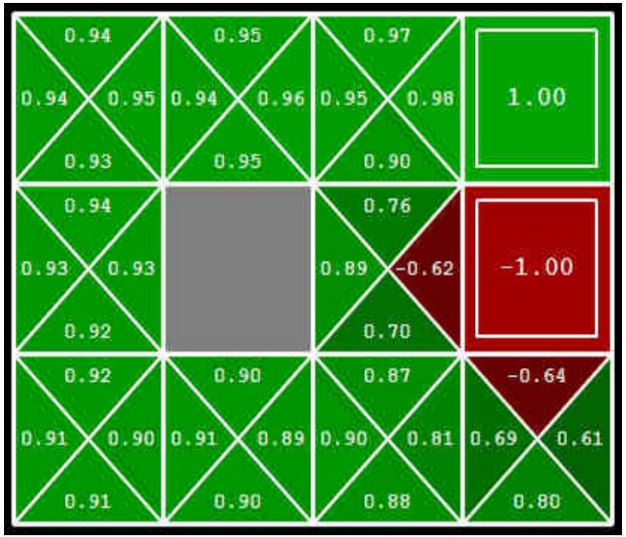


- negative reward for same action > ~100 times in a row
- reward proportional to state[35]
- norm reward = $(R_{(t+1)}) / S_t[35]$ - always almost the same for level(does not depend on actions)
- hypothesis on forum that bbox is stock market. State[35] is portfolio, reward is change in portfolio, action 1/2/3 - buy/sell/no_act, action 0 - ?. other states - financial indicators(that are fixed for levels). low probability of action - slip; norm_reward is financial instrument price

Part B: 5vision solution

B. Q-learning basics

Q- learning:
build/learn a function Q(S,a, w) that tells you how good is action a in state S; w- parameters of function. How good means how many reward will you get in future



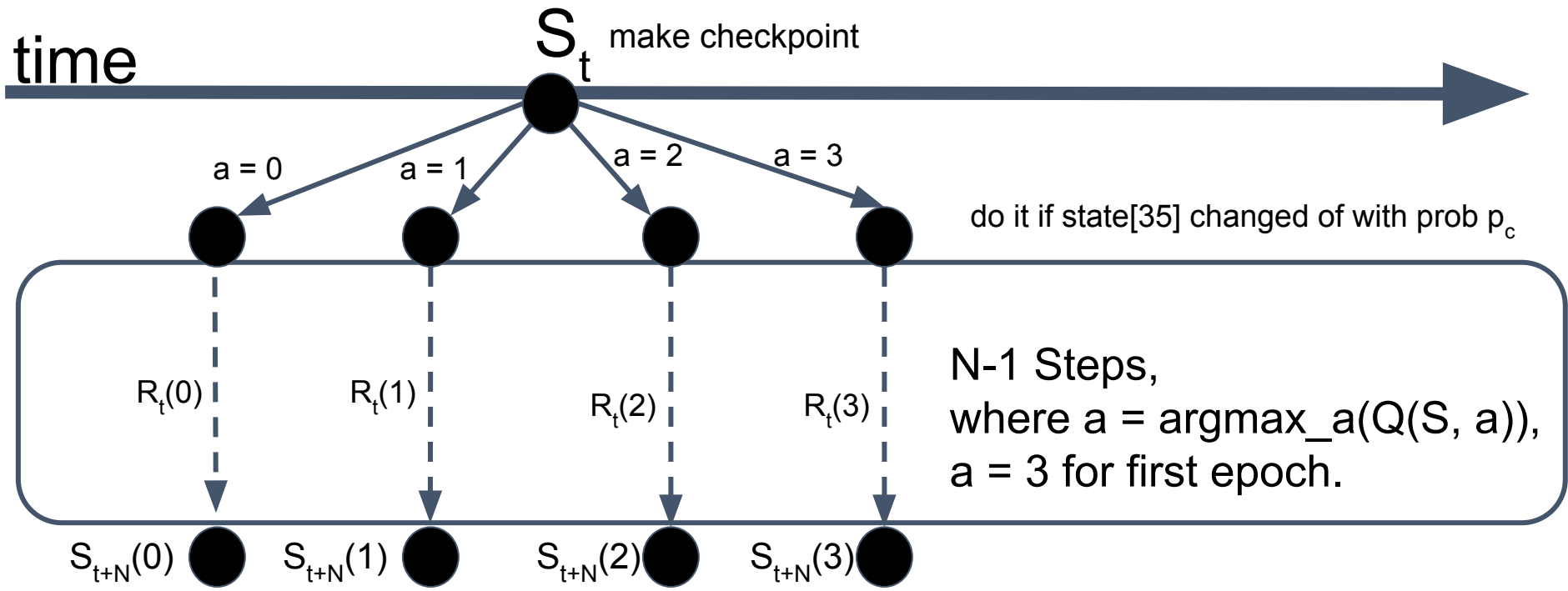
Training Q function (1 step Q learning)

$$J(w) \approx E_{\pi}[(R_t + \gamma max_m \hat{Q}(s_{t+1}, a, w) - \hat{Q}(s_t, a_t, w))^2]$$

When you have such function choose actions as:

$$action = argmax_a \hat{Q}(s, a, w)$$

B. Q-learning for blackbox. Rollout for target calculation



$Y_t(i) = \text{sum}(R_t(i)) + \text{max\_a}(Q(S_{t+N}(i), a)$ (ignore last term for first epoch)

B. Q-learning for blackbox. Policy iteration algorithm

Algorithm

Repeat num_epochs:

Play episode (whole train level)

Get training data S_t, $Y_t(i)$, i =0..3, t = 0...1 200 000

Update w of Q(s,a, w)

Details:

for first epoch choose random action, then argmax_a(Q(S,a))

Policy iteration worked better than asynchronous updates, replay memory, online update.

B. Q-learning for blackbox. Models

Model	Model type	N (rollout steps)	p_c (prob. sampl.)	Comment
reg1	square: 1,2 intr: 1,4,11,12,13	50	0.7	take average weights, retrain separate regr for state[35] ==0, state [35] >0, state[35] <0
reg2	square: 1,2, 14, 16	50	0.7	ake average weights, retrain separate regr for state[35] ==0, state[35] ==0.1, state[35] ==- 0.1, state[35] >0.1, state [35] <-0.1
reg3	square: 0, 1, 2, 9, 10 intr: 4, 1 ,12, 13, 15, 18, 20	50	0.7	take average weights, retrain separate regr for state[35] ==0, state [35] >0, state[35] <0
nn1	1 hid layers: 100 neurons, tanh	50	from 0 to 1 in 10 epochs	50 epochs, take average weights after 30 epochs
nn2	1 hid layer: 16 neurons, tanh	20	from 0 to 0.6 in 10 epochs	50 epochs, take best weights

for reg: square - add square of this state component; intr - multiple this state component on state[35]
for reg - separate regressions for each action, for nn - 4 out neurons, one for each action

B. Final solution

1) **Ensemble of 5 models**
   - initial weights of all model equal to 1
2) **Online learning during evaluation**
   - upload data from test and train level (states and targets for regr1, when state[35] changed)
   - collect states and targets(Yi) on evaluation level;
   - every 200k steps train regr1 and update weights as W = 0.98*W + 0.02 *Wnew;
   - every 200k steps increase weights of regr 1 in ensemble by 0.1;
3) **Restrict actions**
   - if state[35] > 0.5: action 2
   - if state[35] < -0.5: action 1

B. Summing it up

1) n-step Q-learning with large n (20/50)
2) Policy iteration (learn and update policy playing whole level)
3) Take states for learning where state[35] changed
4) Use checkpoints to get data for all actions
5) Online training
6) Restrict actions for high values of state[35], as reward proportional to it
7) cython for fast rollout(sgemm func)/lasagne for training
8) feature engineering (with lasso) for regressions

B. Results

5th place on validation level

Уровень	reg1	reg2	reg3	nn1	nn2	final
train	3081	3642	3547	3325	2735	3536
test	3536	3641	4196	3186	3030	3867
valid	2757	540	1322	2512	2060	3097

code is avaliable here: https://github.com/5vision/blackbox

B. Other ideas

1. gamma < 1, scores were little higher - but were busy with other ideas to retrain all models
2. pca whitening - same results
3. recurrent NN, 1 hour before final submission found little bug in gru - looked promising
4. Lasso/elastic net for regression - worse result
5. adding history - no effect
6. deeper networks - worse result
7. train on both test and train data - overfitting for test
8. Dueling architexture - same result
9. Double Q-learning - same result
10. Online asynch. learning, replay memory - worse result

Part C: Overview of other participants solutions

Part C: Insight: 1 place

Algorithm:

1) Sample states from the environment using the current bot and some noise;
2) Score each of the 4 actions for them: make an action, let the current bot play for 100-500 steps (I found that actions have little effect beyond this), compute the cumulative reward
3) Make a dataset (or a batch), and train a new bot (or update the current one) to tell which turn is better.

Models:

2-3 layer neural networks: 100 neurons in each layer, relu activation functions, 0.3 dropout in the first layer; 4 linear outputs - one per action.

Key components:

1) **Baseline bot** as initial agent
2) **Pairwise loss:** 4 outputs induce 6 pairs, score difference is the weight, sigmoid of the difference of the outputs is the probability that one actions is better than the other, the sum of the cross-entropy losses is the loss. **Score 2150.**
3) **Noisy strategies** (apply a bot for random(N) iterations, make some random actions, score the state) - **Score 2750.** Exponential distribution with mean 3 – got **score 3100**
4) **Ensemble** of 2 models **Score 3300**, **baseline bot** to ensemble **Score 3550**
5) **Train** some models on **train** and some **on test . Score 3750**
6) **Add previous action** to state. **Score 3950**
7) Only **2 layer networks. Score 4048**

Part C: grmel89: 2 place

Algorithm(genetic algorithm):

1) Initial population based on baseline regression
2) Parents selection
2) Cross-over
3) Mutation
4) Selection
5) Iteration < 200: go to step 2.

Models:

Logistic regression (baseline with softmax)

Key components:

1) Train in test and train data
2) Probabilistic selection of level (train 70%, test 30%)
3) Initial population - baseline agent
4) Use random start

Part C: wrwrwr: 3 place

Algorithm(genetic algorithm/local diminishing search):

1) Start with random parameters drawn from a normal distribution (possibly selecting the best set from a couple of tens or hundreds of samples).
2) For 100-100000 steps redraw a few array entries and combine them with the old entries.
3) Replace old weights with new ones at the beginning or weighted average at the end, if score was improved

Models:

linear regression

Key components:

1) Beta distribution for number of parameters to change (with peak between 6-9)
2) Make the coefficients for state[35] bigger
3) Adding weights for the difference between the current and the last state
4) "belief" bot (one that imagines that the state has an additional component, and uses it to evaluate actions in the same manner that it uses the real features, even though it has no idea how to properly update the belief)

Part C: VictorGNC: 4 place

Algorithm(genetic algorithms):
2 network - ElitistCMA,
2 network - LMCMAlocal

Models:
neural networks, 36-64-4 и 16-4

Key components:
1) Ensemble of four models
2) Maximize minimum score on sublevel of level (each level consist of about 10 sublevels)
3) Minimize L1 norm of weights

$$\text{Maximize } \min(sublevel\_score(w)) - \eta \|w\|_1$$

Links to solutions

1. **Insight:**

   https://github.com/pv-k/bb

2. **wrwrwr:**

   https://github.com/wrwrwr/blackbox

4. **VictorGNC:**

   https://github.com/gncgroup/BlackBox-Solver

5. **5vision:**

   https://github.com/5vision/blackbox