Automated optimization of a course of actions in a stochastic environment

Thomas Vicente

Introduction

First sentence: ~philosophical. A complete exercise would be to build an autonomous entity that is able to take long term oriented decisions based on information coming from the phisical reality.

This entity needs two abilities: learning from its own actions and recognising the current environment's patterns. The first ability is typically made possible by a Reinforcement Learning (RL) mechanism. The second ability is done by regressors or classifiers powerful enough to adapt to the entity's sensorial features. In particular, if the entity is equipped by a visual sensor, neural networks might be a good idea.

RL emerges easily from other Machine Learning techniques as the training is done by acting. The learner evolves in particular environment and records what actions lead to a maximal gain. In most cases the goal is to maximise a long term gain by taing a correct course of actions. These policies can be more or less explicit. An RL learner is particularly dependent on a state signal that succeeds in retaining all relevant information ; this state is said to be Markov. For example, a checkers position (the current configuration of all the pieces on the board) would serve as a Markov state because it summarizes everything important about the complete sequence of positions that led to it. Much of the information about the sequence is lost, but all that really matters for the future of the game is retained.

DO I DO ON OR OFF POLICY? Policy related to Markov decision process

Examples of regressors/classifier "recognising the real world" Using an approximator has two benefits: ability to provide an evaluation of an infinite set of states and recognise the underlying states (equivalent for the purpose of a particular task) when the signal is imperfect, which is our case with our study.

Tic tac toe, a classical RL application, is also a good example for the principle studied. I present an application of the approchaed principle in the second part.

Litterature review

The closest algorithm from our principle is in 9.4 in Sutton (2012). But the proposed principle differs from it. First, we consider the possibility of same underlying states with imperfect signals (in a handwritten game, this is due to the differences in handwritten letters every time a human plays). Second, we do not use Temporal differences but a discount method.

Deep Q networks (DQN) are similar to the proposed principle as they store long term data. This is the so called "Experience replay": when training the network, random minibatches from the replay memory are used instead of the most recent transition. Such efficient implementation makes the DQN very adaptive to different tasks, as demonstrated for the learning of multiple arcade games while using the same network architecture.

General principle

Explain how it works in sentences

Alternative discount

Table of all possible values from discounted equation (about 12 values I guess)

quick review of different regressors

In the algorithm description, we try to get a maximum of generality and thus use the term "regressor".

Seems that "discount post signal" is the particularity of model. Give empirical insights why it works/is efficient. Good point: each step takes a value, not only final ones (can represent with a colored gradient picture)

Explanation of some elements:

- Ω can be set deterministically. In the context of a game, there is a well determined space of actions. It also can be infinite. A mobile robot can explore the real 3D world in almost infinite ways.
- The signal value function assumes the existence of one or multiple sensors, or the manual assignement of a value. It is conditional on the current environment and takes positive value if a "gain" is sensed, and negative value if a "pain" is sensed.
- The minimization process is a tedious part of the algorithm. When dealing with unstructured features, we might want to use a neural network-type regressor. In that case, W, the set of the hidden layers' weights is initialized with the values of the previous iterations. The optimal W^* should converge as X grows if there are patterns in the stochastic processes.

Remark on the function of the regressor's role

The regressor's task is not to predict accurately, even though it is closely related, but rather to give the "real value" of an action.

Algorithm 1 General version

Initialized:
- X: empty m-columns feature matrix
- y: empty target vector
- X': empty m-columns simulated states matrix
- $w^* = w_0$: random weights for the regressor
- $\boldsymbol{\gamma} \in [0, 1]$ discount rate
- $\Delta = 0$: number of actions

Other variables:
- $f()$: regressor function
- m: number of features
- $p_t(x^*)$: stochastic process, function of the last existing chosen state
- $x_t(p_t) \in \mathbb{R}^{\mathrm{m}}$: current state vector
- $\Omega_t(x_t)$: set of possible actions
- $s(x_t) \in \mathbb{R}$: signal value triggered by the modified state

WHILE the learning process is on:
- record x_t
- $X' :=$ empty matrix
- **FOR** x' in $\Omega_t(x_t)$:
  - append x' to X'
- With probability $1 - \epsilon$
  - $x^* := \arg\max_{x' \in X'} f(w^*, X')$
- With probability ϵ
  - $x^* :=$ random sample $x' \in X'$
- append x^* to X
- **IF** $s(x^*)$!= 0 :
  - **FOR** δ in 1:Δ:
    - $value = \gamma^{\delta-1} s(x^*)$
    - append $value$ to y
  - $w^* := \arg\min_w(l(f(w, X), y))$
  - $\Delta := 0$
- **ELSE**:
  - Δ += 1

Analysis of why RL+NN converge, why it works

It needs to have a good balance between gains and pains Find some equations, ask Gabor then

Cool remark

sometimes the triggering a new signal can take time. But robot can still look for best discounted s(xˆ*). Wecan consider that the robot "discovers" new objectives in that context. True in general RL

Application

Explain game "Long term" (3-5 actions) strategy matters. Discount acts here The environment contains rivalry. Another, more classical, robot.

Talk about tweaks

generation of images other guy for selfplay... rotations to go faster than Other learn not after each game

Remark on the function of the regressor's role

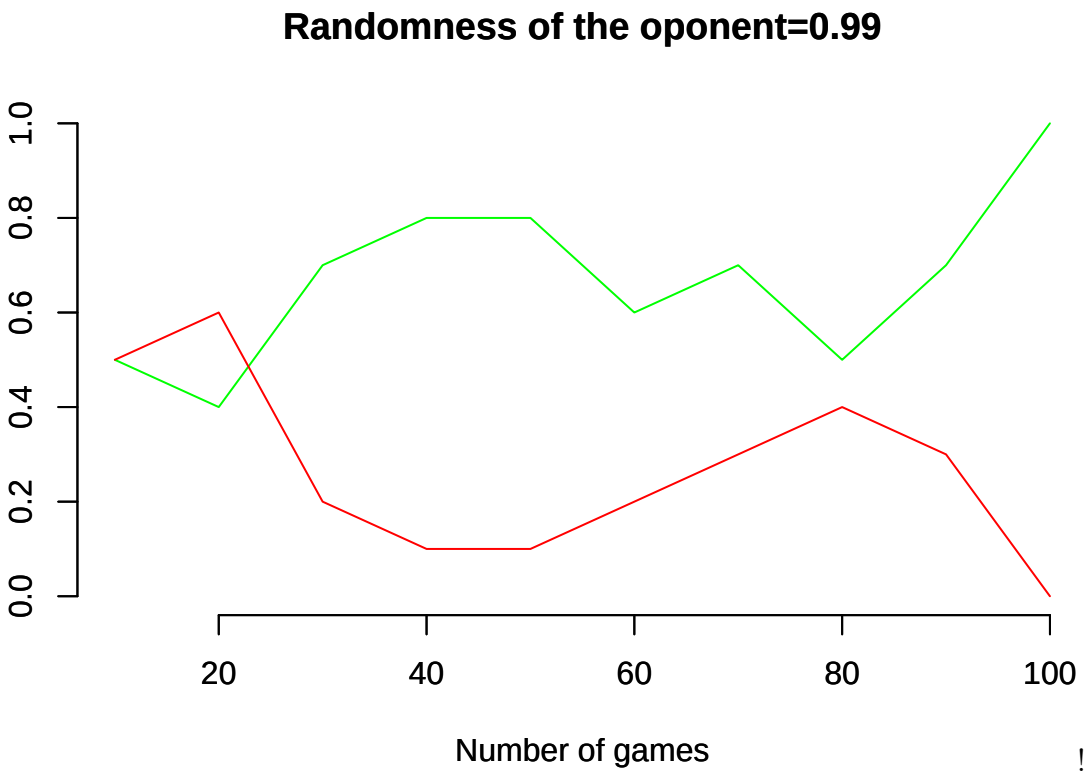
Two intermediary movements can ultimately lead to both winning and losing. Such states will get a non-tendencial value as X grows, which is a good thing.

why chose NN. explain how it works. choice of architecture

Analysis of game

Do some graph based on data.txt's target variable, observe time effects, wins evolution Proof of convergence

The following plots show the evolution the robot's performance during the learning phase. Recall that the robot updates its parameters every 100 (MIGHT CORRECT) games. The green line corresponds to the proportion of won games for the last 100 games The red line corresponds to the proportion of lost games for the last 100 games



!

The last plot shows the average number of moves before winning for the last 100 games.

Notice after 100 games, able to play "against" image if move easy, like last action

Results/other applications

References

http://people.inf.elte.hu/lorincz/Files/RL_2006/SuttonBook.pdf

http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Teaching_files/FA.pdf