Java Regret Code Documentation

Alan Dix, Nov 2021

See: https://www.alandix.com/academic/papers/regret-2021/

Adjustable Parameters

Skinner Learning Parameters

```
double gain = 1.0;
```

factor multiplied by positive or negative outcomes when determining increment for weight of stimulus-response

```
double weightPower = 2;
```

when deciding between responses, the weights of each are raised to this power to give probabilities of selection. A value of 1 means a response with twice the weight is twice as likely to be chosen (linear). Higher values take the learner closer to 'winner takes all', and as optimisation values greater than 10 are treated as 'winner takes all', that is the largest weight wins.

```
double badBias = 1.0;
```

psychology experiments often show that bad things are 'more bad' than an apparently equivalent good thing. Setting badBias to larger values makes bad effects be taken more seriously than good ones (e.g. if badBias is 2, then one bad effect counts twice as much as an equivalent good one).

```
double unseenWeight = -50;
```

this gives an initial weight for as yet untried responses
a value of -50 is risk averse - doesn't like new things, sticks and with the old unless really bad
in contrast +50 would be a risk taker - likes to try new things

```
double maxPositive = 100;
double maxNegative = -100;
```

maximum and minimum values of weights
non-linear scaling is applied at each step to ensure the weights stay in this range

Regret Learning Parameters

```
double POS_REGRET_FACTOR = 1;
double POS_NO_REGRET_FACTOR = 1.0;
double NEG_REGRET_FACTOR = 1;
double NEG_NO_REGRET_FACTOR = 0.5;
```

Classes

'Main' classes

```
  Main.java
      driver, does either single run of each kind of learning, or does
      a series with the same parameters, but different card packs
  RunRegret.java
      applies various types of 'Thinkers' to various kinds of 'Game'
```

basic stimulus response learning

```
    ConditionedLearner.java
          - interface describing simple stimulus response style of learning
            can be asked to give a response for a given stimulus and then
            afterwards can be asked to condition itself by giving a score
            (goodness/badness) to the effect of the response
    . Stimulus.java
    . Response.java
          - interfaces used to represent abstract stimulus and response
            kinds by ConditionedLearner
    RandomResponse.java
          - implementation of ConditionedLearner.java that simply returns
            random response each term - that is no learning.  This is used
            to give a baseline for what might be considered effective learning
    Skinner.java
          - skinner-like learner that simply stores against every stim-resp
            pair a  'how good I feel about it' which is updated depending
            on how good/bad it ends up being.
            This can support (via run time options) multiple methods for
            choosing the response from a simple 'winner takes all' to
            variants of 'better is more likely'.
    . StimRespPair.java
          - utility class used to give stimulus response pairs a single
            hash so that they can be used as keys
```

potentially more complex learning

```
    Thinker.java
          - a Thinker is like a ConditionedLearner, but, in addition to
           the score of the actual response, it is given and 'Afterwards'
           object which can be used by the thinker to probe potential
           alternative outcomes.
    . Afterwards.java
          - interface to represent an abstract state of the world after
            a response has been given
    SimpleThinker.java
          - an implementation of Thinker that simply wraps a ConditionedLearner
            and ignores the additional 'Afterwards' information
    RegretThinker.java
          - adds regret to a simple conditioned response
    . Replayer.java
          - used by RegretThinker to try all possible moves after the game
            has finished to work out what would have been best
```

games to play

```
Game.java
     - interface giving abstract view of different games
       a Game object can be asked to generate a new randomised
       starts state (instance of Before interfaces), given a Before
       instance an generate possible play moves that could be performed,
       and given a Before state and a Play move give the
       corresponding effect, and After object.
. GameStimulus.java
. GameResponse.java
. GameAfterwards.java
     - wrapper classes to make game Before, Play and After objects
       act like Stimulus, Response and Afterwards
. GameReplayer.java
      - implementation of Replayer specialised to game
SimpleGame.java
      - sort of pontoon/blackjack without the bank, the player gets
        a single card and has to decide whether to stick or twist.
        The player's score is the sum of the cards if a twist and
        the first card if stick, but looses a penalty of the cards
        go over a limit ('bust').
PontoonGame.java
      - with banker, normal blackjack rules except only one card
        dealt initially to player and bank, the maximum card value
        is usually something small (e.g. 3) and the bust limit
        similarly small (e.g. 4)
```

some utility classes

```
CardPack.java
      - pack of cards, can be asked to shuffle itself, but the pack
        can be cloned to allow the same card sequence to occur in
        different conditions
Util.java
      - as it says!
```