`Alsim`, an associative learning simulator

Stefano Ghirlanda

Very preliminary version of June 16, 2018

1 Introduction

1.1 Scope

`Alsim` is an associative learning simulator for models that can be cast in the "kernel machine" framework described in Ghirlanda [2015, 2018]. These include the Rescorla and Wagner [1972] model, Pearce's (1987) "configural" model, Brandon et al.'s (2000) "replaced elements" model, and others. `Alsim` is a trial-based simulator in which each trial is constituted of an experience with one or more stimuli followed by a reinforcement value. The simulator tracks the values of associative strengths between stimuli and (depending on your theoretical leaning) responses or reinforcers.

`Alsim` is not a real-time simulator, and it does not support internal structure of trials, such as sequences of stimuli or responses.

1.2 Overview

`Alsim` is programmed in R [R Core Team, 2017]. A model is created with the `newModel` function, which accepts two arguments: a generalization function and a learning rate.[1] See Section 4 for an explanation of generalization functions. Once created, a model can be trained with the following functions:

`trial:` performs a single trial with a stimulus.

`phase:` can perform multiple trial with multiple stimuli.

`experiment:` can train multiple subject groups, each group undergoing multiple training phases.

After training, associative strengths can be calculated with the `V` function and visualized with the `Vplot` function. See Sections 2 and 3 for examples.

[1] Extension are planned in which the learning rate can be specified per stimulus and per λ value. Other extensions will enable changing learning rates through learning, for example to model attention.

1.3 Installation

`Alsim` is at https://www.github.com/drghirlanda/alsim. You can install it in several ways:

- Using `devtools` in R: `devtools::install_github( "drghirlanda/alsim/alsim" )` (yes, you need the `alsim` twice).
- Download the whole `github` project and then run `R CMD INSTALL alsim` from the top-level `Alsim` directory.
- Download the `.tar.gz` file from `github` and then run `R CMD INSTALL alsim_X.Y.tar.gz`, where `X.Y` is the version number.

This assumes you have permissions to install. On a personal Linux machine, you can probably start R as `sudo R` and things will work.

2 Examples

2.1 Acquisition and external inhibition

The following code simulates Pearce's (1987) model in acquisition trials with a stimulus *A* followed by an external inhibition test with a compound, *AB*, of *A* and a novel stimulus *B*:

```
1 Mod <- newModel( pearce, 0.1 )       # create a pearce model
2 A  <- c(1,0)                         # create stimulus A
3 AB <- c(1,1)                         # create stimulus AB
4 Mod <- phase( Mod, list(A=c(30,1)) ) # train 30 trials with lambda=1
5 VA <- V( Mod, "A", 0:30 )            # calculate VA across trials
6 VAB <- V( Mod, "AB" )                # calculate VAB (at trial n)
```

Here is some more explanation by line number:

1. `newModel` creates a model using a generalization function (`pearce`) and learning rate (0.1).
2. Each stimulus is a vector, in this example with two elements. In `A`, only the first one is active, signifying the presence of *A*.
3. In `AB`, both elements are active. Thus the second element signifies the presence of *B*.
4. The `phase` function runs training trials. A phase is a named list with names corresponding to stimuli. Each list element is a 2-element vector. The first element is the number of trials with that stimulus, the second the reinforcement value. Thus `list( A=c(30,1) )` specifies to train *A* for 30 trials, with reinforcement value of 1. **Note:** `phase` does not change the model argument. It returns a copy that has been modified according to the training.
5. The `V` function calculates associative strength. It also accepts stimulus names as strings, and trials at which associative strength should be calculated.
6. If no trial argument is given to `V`, the function returns the associative strength after the last trial. Hence `V( Mod, "AB")` calculates the associative strength of *AB* at a test following acquisition.

2.2 Summation

The following example shows how to mix trials with different stimuli. It is a simple summation experiment with the replaced elements model:

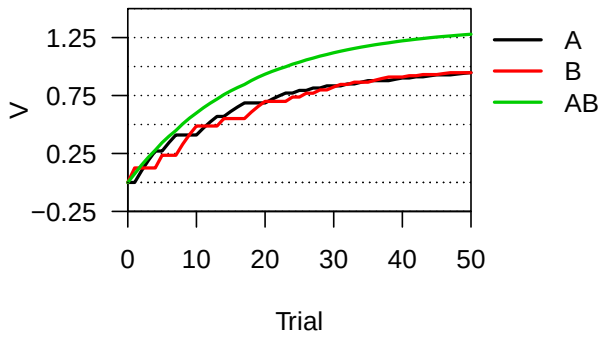
```
alsim.options$rem$r <- 0.25 # set replacement factor
Mod <- newModel( rem, .1 )  # create a rem model

A <-  c( 1, 0 ) # create stimuli A, B, and AB
B <-  c( 0, 1 )
AB <- c( 1, 1 )

Mod <- phase(      # train the model with 50 presentations
    Mod,           # of A and 50 of B, all rewarded with a
    list(
        A=c(50,1), # 50 trials with A and reinforcement = 1
        B=c(50,1)  # 50 trials with B and reinforcement = 1
    )
)

# visualize results:
Vplot( Mod, c("A","B","AB"), 0:50 )
```

The last line plots the associative strengths of the given stimuli. Trial 0 represents the state of the model prior to any learning. Any stimulus can be plotted, even if it was not part of training. In this case, we plot `AB` as well as `A` and `B` to assess summation. The `Vplot` produces a plot such as this one:



The lines in the figure are somewhat jagged because they represent the outcome of a single simulation. You may be used to seeing smoother lines representing theoretical learning curves or averages of many simulations. The `experiment` function detailed in the next section can run multiple subjects, after which the same `Vplot` function can be used to visualize average learning curves.

`Vplot` is meant for quick visualization and not for publication-quality output, for which R has plenty of options.

3 Experiments

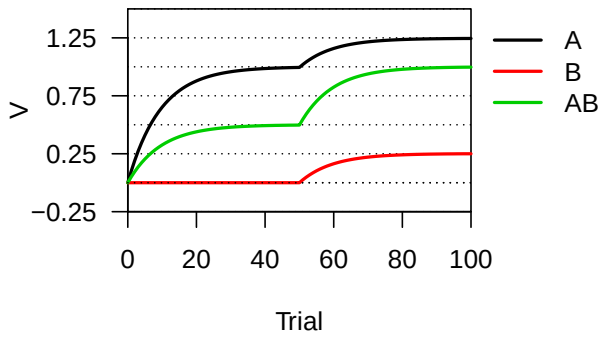
Training phases can be grouped together in experimental designs, which are lists of experimental groups, each experimental group being itself a list of phases. Phases are in the format seen above. For example, Kamin's (1969) classic blocking design can be expressed as:

```
blocking <- list(
    experimental=list(
        phase1=list( A=c(50,1) ),
        phase2=list( AB=c(50,1) )
    ),
    control=list(
        phase2=list( AB=c(50,1) )
    )
)
```

An experiment can be run on multiple (simulated) subjects with the `experiment` function. For example:

```
A <- c(1,0)
B <- c(0,1)
AB <- c(1,1)
model <- newModel( pearce, 0.1 )
results <- experiment( model, blocking, 20 )
Vplot( results$experimental, c("A","B","AB"), 0:100 )
```

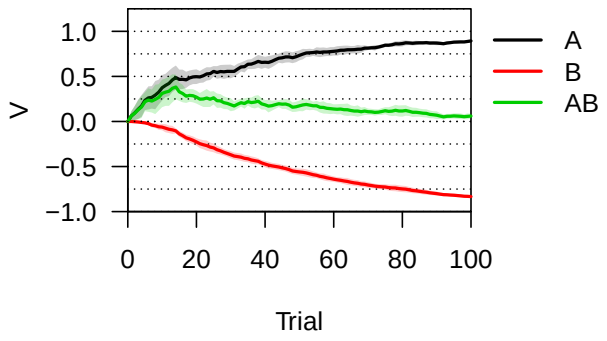
where the last argument to `experiment` is the number of subjects in each group. Aggregate results can be computed with `V` and `Vplot` as before. For example, the last line of the code above produces this plot:



In the case of blocking, all experimental subjects undergo the same sequence of trials, hence there is no variability across subjects. In experiments that mix different stimuli, however, `Alsim` generates different trial sequences for each subjects. In this cases, `Vplot` visualizes standard deviations as well as average associative strengths. For example, the following is a simulation of a feature negative discrimination with the Rescorla and Wagner [1972] model (obtained as the replaced elements model with no replacement):

```
A  <- c(1,0)
B  <- c(0,1)
AB <- c(1,1)
model <- newModel( rem, 0.1 )
alsim.options$rem$r <- 0
featureNegative <- list(
    S=list(
        phase1=list(
            A=c(50,1),
            AB=c(50,0)
        )
    )
)
results <- experiment( model, featureNegative, 10 )
Vplot( results$S, c("A","B","AB"), 0:100 )
```

And the resulting graph is:



4 Generalization functions

Generalization functions are at the heart of `Alsim`. A generalization function tells us how the associative strength of a stimulus generalizes to other stimuli. For example, if $g(A,B) = 1/2$, then half of V_A generalizes to B. Introduced by Pearce [1987] for his "configural" model, generalization functions can also be considered for "elemental" models. For example, in the Rescorla and Wagner [1972] model, $g(A,AB) = 1$ because V_A generalizes completely to AB, under the hypothesis that AB contains all of the stimulus elements in A. In the replaced elements model, on the other hand, $g(A,AB) = 1-r$, where r is the replacement factor, because only a fraction $1-r$ of A's elements are present in AB.

To make a long story short, Pearce's (1987) configural formalism can be generalized to apply to elemental as well as configural models [Ghirlanda, 2015, 2018] by thinking in terms of generalization functions.

5 Stimulus representations

Generalization function will typically return different values for different stimulus representations. Stimulus representations are thus an integral part of modeling. For example, `pearce` (or any other generalization function) will return zero if A and B have no common elements:

```
A <- c(1,0)
B <- c(0,1)
pearce( "A", "B" )
```

```
[1] 0
```

But it will return non-zero values when common elements are assumed:

```
A <- c(1,0,1)
B <- c(0,1,1)
pearce( "A", "B" )
```

```
[1] 0.25
```

6 Models

`Alsim` comes with an implementation of the following models:

`pearce` is the model in Pearce [1987, 1994].

`elemental` is a generic "elemental" model in which $g(X,Y) = \sum_i X_i Y_i$. This is the Rescorla and Wagner [1972] model with general stimulus representations, including the possibility of unique cues, common cues, etc. These cues have to be set explicitly in stimulus vectors.

`rem` is the replaced elements model in Brandon et al. [2000], Wagner and Brandon [2001], Wagner [2003, 2008]. Setting the replacement rate to zero yields the Rescorla and Wagner [1972] model. Note that `rem` takes into account replacement without having to construct stimulus representations with replaced elements. It is be possible to use `elemental` to model the replaced elements model, but one would have to construct appropriate stimulus vectors explicitly (see Section 8).

7 Making your own

To add a model to `Alsim`, all you need to do is write a generalization function. The generalization function must take as input two stimulus names, as strings, and must return a single numeric value. There are two possible approaches. The first is to operate on vector representations of stimuli. The second is two operate on stimulus names only. As an illustration of the first approach, imagine that you want to add a silly model in which generalization depends only on the first component of each vector. If they are equal, generalization is 1, otherwise it is 0. This is the way to do it:

```
1 silly <- function( X, Y ) {
2     vX <- name2vec( X )
3     vY <- name2vec( Y )
4     if( vX[1] == vY[1] )
5         g <- 1
6     else
7         g <- 0
8     g
9 }
```

Here, lines 2 and 3 retrieve the vector associated with the names `X` and `Y`. To see `silly` at work, we first define some stimulus vectors:

```
1 X <- c(1,1)
2 Y <- c(1,0)
3 Z <- c(0,1)
```

After which we can try:

```
1 silly( "X", "Y" )
```

```
[1] 1
```

or:

```
1 silly( "X", "Y" )
```

```
[1] 1
```

Note that `silly` fails if the vector is not defined:

```
1 silly( "X", "undefined stimulus name" )
```

```
Error in get(X, pos = 1, mode = "numeric") :
  object 'undefined stimulus name' of mode 'numeric' was not found
```

The second approach to writing generalization functions does not relying on defining vectors, but rather it assigns generalization values to named stimuli directly. If you want/need to take this route, here is a template:

```
1 manual <- function( X, Y ) {
2     known <- c("A","B","AB")
3
4     if( !( X %in% known ) || !( Y %in% known ) )
5         stop(paste(
6             "use only known stimuli:",
7             paste(known,collapse=" ")
8         ) )
```

```
    g <- matrix( 0, nrow=3, ncol=3 )
    rownames(g) <- known
    colnames(g) <- known
    g["A","A"] <- g["B","B"] <- g["AB","AB"] <- 1
    g["A","B"] <- g["B","A"] <- 0
    g["A","AB"] <- g["AB","A"] <- 1/2
    g["B","AB"] <- g["AB","B"] <- 1/2

    g[X,Y]
}
```

This defines generalization only between `A`, `B`, and `AB`. For example:

```
manual( "A", "AB" )
```

```
[1] 0.5
```

But:

```
manual( "A", "C" )
```

```
Error in manual("A", "C") : use only known stimuli: A B AB
```

8 Issues

- `rem` is slow because it computes common pairs every time. It would be better to have a `remify` function to modify vectors and then use `elemental` in the model. The function would need to know the maximum number of stimulus pairs, and it would need a way to index them. Probably `combn(n,2)` where n is the size of the original vectors. Then we can add $n(n-1)/2$ components to represent pairs and set them to r while we set to $1-r$ the components that correspond to the elements of each pair.

References

Susan E. Brandon, Edgar H. Vogel, and Allan R. Wagner. A componential view of configural cues in generalization and discrimination in Pavlovian conditioning. *Behavioural Brain Research*, 110: 67–72, 2000.

Susan E. Brandon, Edgar H. Vogel, and Allan R. Wagner. A componential view of configural cues in generalization and discrimination in pavlovian conditioning. *Behavioural Brain Research*, 110: 67–72, 2000.

S. Ghirlanda. On elemental and configural theories of associative learning. *Journal of Mathematical Psychology*, 64-65:8–16, 2015.

S. Ghirlanda. Studying associative learning without solving learning equations. Preprint. DOI: 10.6084/m9.figshare.6267047.v1, 2018.

L. J. Kamin. Predictability, surprise, attention, and conditioning. In B. A. Campbell and M. R. Church, editors, *Punishment and aversive behavior*, pages 279–296. Appleton-Century-Crofts., New York, 1969.

J. M. Pearce. A model for stimulus generalization in Pavlovian conditioning. *Psychological Review*, 94(1):61–73, 1987.

J. M. Pearce. Similarity and discrimination: a selective review and a connectionist model. *Psychological Review*, 101:587–607, 1994.

J.M Pearce. *An introduction to animal cognition*, volume 1. Lawrence Erlbaum Associates, 1987.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2017. URL https://www.R-project.org/.

R. A. Rescorla and A. R. Wagner. A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. In *Classical conditioning: current research and theory*. Appleton-Century-Crofts, 1972.

A. R. Wagner. Context-sensitive elemental theory. *Quarterly Journal of Experimental Psychology*, 56B:7–29, 2003.

A. R. Wagner. Evolution of an elemental theory of Pavlovian conditioning. *Learning & Behavior*, 36(3):253–265, 2008.

Allan R. Wagner and Susan E. Brandon. A componential theory of pavlovian conditioning. In Robert R. Mowrer and Stephen B. Klein, editors, *Handbook of contemporary learning theories*, chapter 2, pages 23–64. Lawrence Erlbaum Associates, London, 2001.