Speaking and listening

Agent-based modelling, Konstanz, 2024

Henri Kauhanen

30 April 2024

❗ Update 7 May 2024

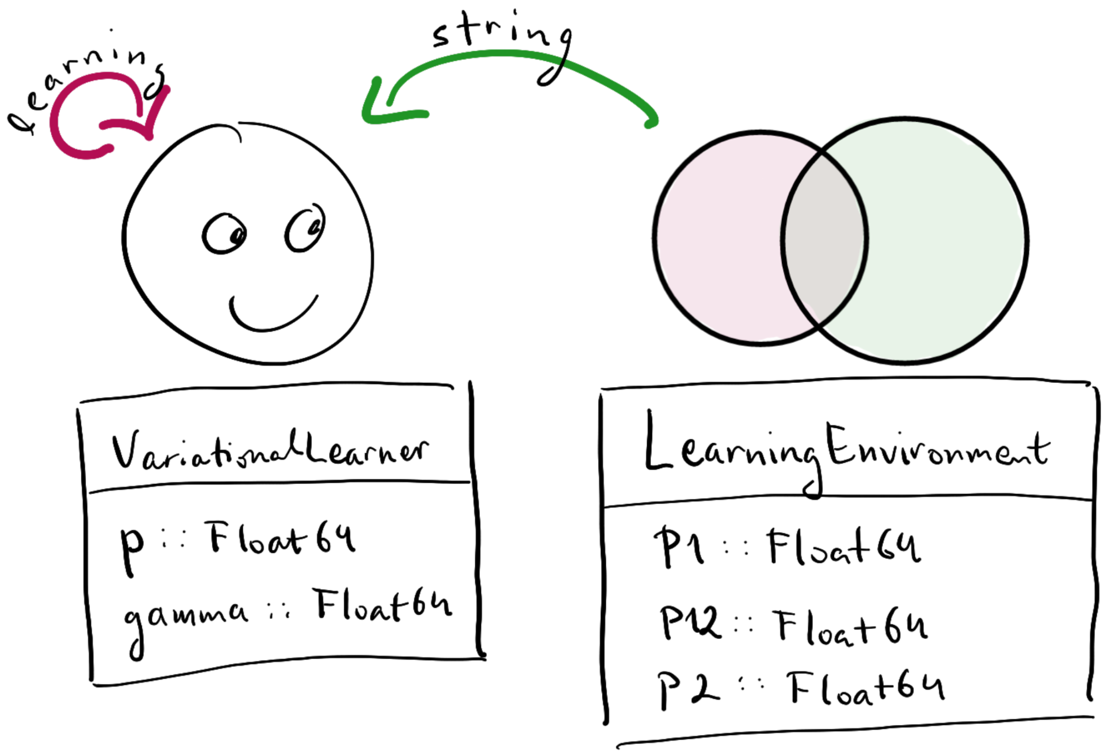
Fixed the link to the homework.

Plan

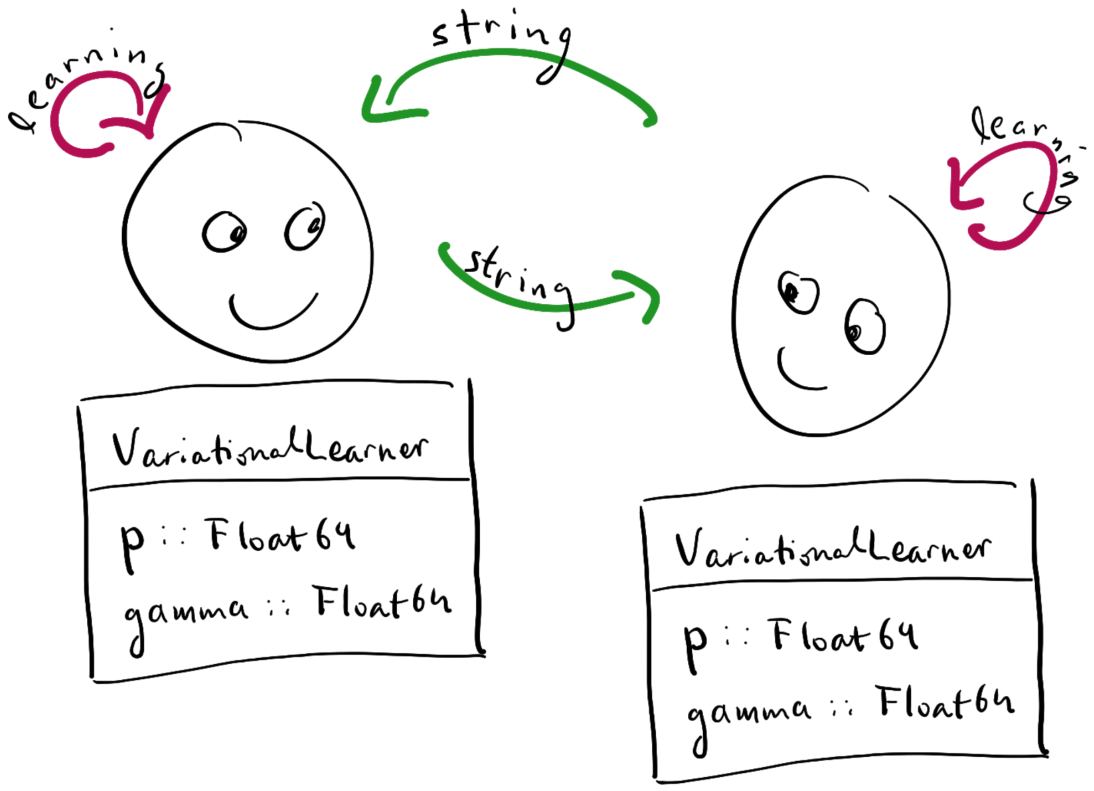
- Last week, we ran out of time
- Better go slowly and build a solid foundation rather than try to cover as much ground as possible
- Hence, today:
    - Finish last week's material
    - Introduce a little bit of new material: implementing interactions between variational learners

Dropping the environment

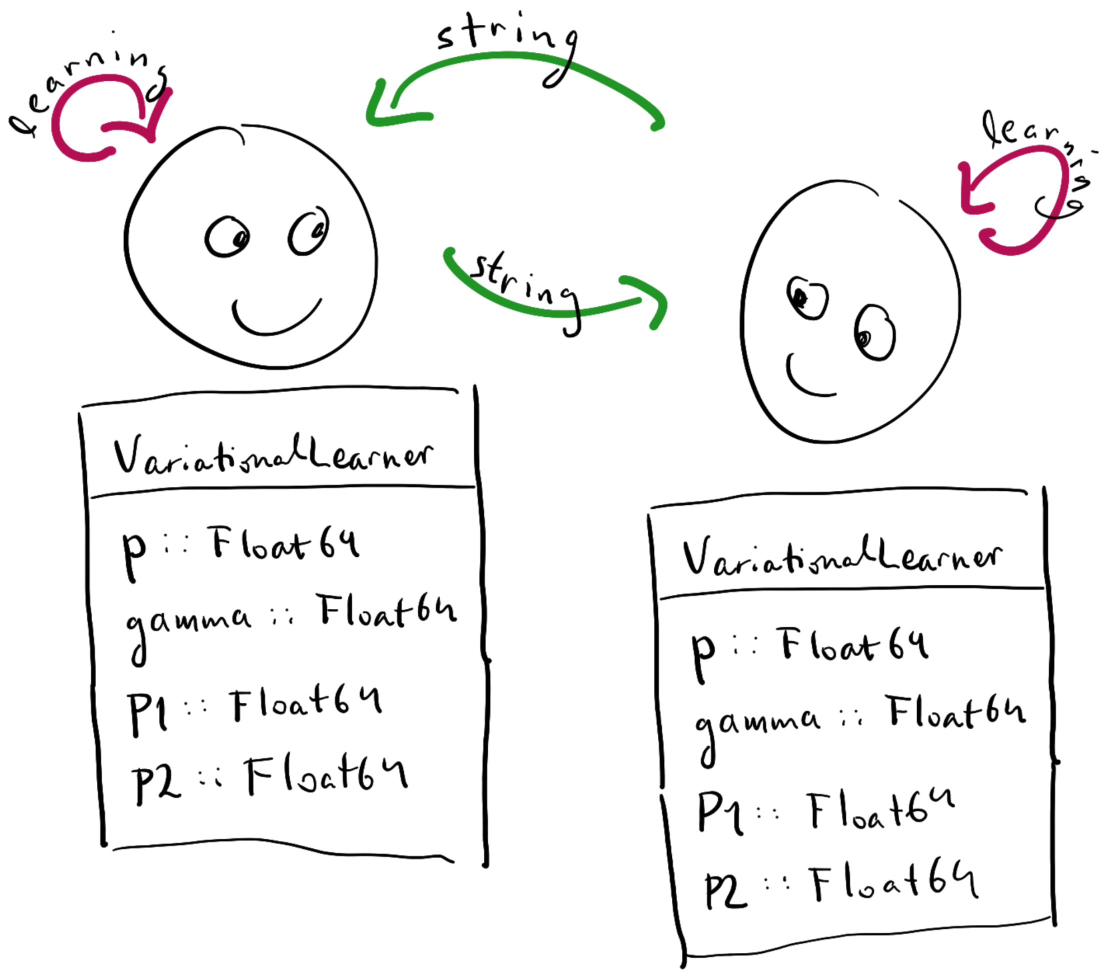
- So far, we've been working with the abstraction of a `LearningEnvironment`:



- We will now drop this and have two `VariationalLearner`s interacting:



- The probabilities `P1` and `P2` now need to be represented inside the learner:



- Hence we define:

```
mutable struct VariationalLearner
  p::Float64      # prob. of using G1
  gamma::Float64  # learning rate
  P1::Float64     # prob. of L1 \ L2
  P2::Float64     # prob. of L2 \ L1
end
```

Exercise

Write three functions:

- `speak(x::VariationalLearner)`: takes a variational learner as argument and returns a string uttered by the learner
- `learn!(x::VariationalLearner, s::String)`: makes variational learner `x` learn from string `s`
- `interact!(x::VariationalLearner, y::VariationalLearner)`: makes `x` utter a string and `y` learn from that string

Answer (speak)

```julia
using StatsBase

function speak(x::VariationalLearner)
  g = sample(["G1", "G2"], Weights([x.p, 1 - x.p]))

  if g == "G1"
    return sample(["S1", "S12"], Weights([x.P1, 1 - x.P1]))
  else
    return sample(["S2", "S12"], Weights([x.P2, 1 - x.P2]))
  end
end
```

```
speak (generic function with 1 method)
```

Answer (learn!)

```julia
function learn!(x::VariationalLearner, s::String)
  g = sample(["G1", "G2"], Weights([x.p, 1 - x.p]))

  if g == "G1" && s == "S1"
    x.p = x.p + x.gamma * (1 - x.p)
  elseif g == "G1" && s == "S2"
    x.p = x.p - x.gamma * x.p
  elseif g == "G2" && s == "S2"
    x.p = x.p - x.gamma * x.p
  elseif g == "G2" && s == "S1"
    x.p = x.p + x.gamma * (1 - x.p)
  end

  return x.p
end
```

```
learn! (generic function with 1 method)
```

Answer (interact!)

```julia
function interact!(x::VariationalLearner, y::VariationalLearner)
  s = speak(x)
  learn!(y, s)
end
```

```
interact! (generic function with 1 method)
```

Picking random agents

- `rand()` without arguments returns a random float between 0 and 1
- `rand(x)` with argument `x` returns a random element of `x`
- If we have a population of agents `pop`, then we can use `rand(pop)` to pick a random agent
- This is very useful for evolving an ABM

Aside: `for` loops

- A `for` loop is used to repeat a code block a number of times
- Similar to array comprehensions; however, result is not stored in an array

```julia
for i in 1:3
  println("Current number is " * string(i))
end
```

```
Current number is 1
Current number is 2
Current number is 3
```

A whole population

- Using a `for` loop and the functions we defined above, it is now very easy to iterate or evolve a population of agents:

```
pop = [VariationalLearner(0.1, 0.01, 0.4, 0.1) for i in 1:1000]

for t in 1:100
  x = rand(pop)
  y = rand(pop)
  interact!(x, y)
end
```

Exercise

Write the same thing using an array comprehension instead of a `for` loop.

Answer

```
pop = [VariationalLearner(0.1, 0.01, 0.4, 0.1) for i in 1:1000]

[interact!(rand(pop), rand(pop)) for t in 1:100]
```

```
100-element Vector{Float64}:
 0.1
 0.1
 0.1
 0.1
 0.1
 0.099
 0.1
 0.1
 0.1
 0.1
 0.1
 0.1
 0.1

 0.09801
 0.1
 0.10900000000000001
 0.099
 0.1
 0.1
 0.1
 0.1
```

```
0.1
0.1
0.1
0.1
```

Next time

- Next week, we will learn how to **summarize** the state of an entire population
- This will allow us to track the population's behaviour over time and hence model potential **language change**
- This week's homework is all about consolidating the ideas we've looked at so far – the variational learner and basics of Julia