model-demo

November 13, 2018

1 Model demo

```
In [1]: from model import Model
        from chunk import Chunk
        import numpy as np
        import matplotlib.pyplot as plt
```

1.1 Creating a model

```
In [2]: m = Model()
```

We can use `print(m)` to print an overview of the model:

```
In [3]: print(m)
```

```
=== Model ===
Time: 0 s
Goal:None
DM:
```

1.2 Setting a goal

Add a chunk to the model's goal buffer. We can specify a chunk name and any number of slots (as a dictionary). Here we first create a chunk with the name "goal-chunk" that has two slots.

```
In [4]: g = Chunk(name = "goal-chunk", slots = {"goal": "count", "current": "two"})
        m.goal = g
```

Check that the goal is added to the model:

```
In [5]: print(m)
```

```
=== Model ===
Time: 0 s
Goal:Chunk goal-chunk
```

```
Slots: {'goal': 'count', 'current': 'two'}
Encounters: []
Fan: 0

DM:
```

1.3 Adding chunks to memory

Here we add some chunks to the model's declarative memory (at t = 0).

```
In [6]: c1 = Chunk(name = "c1", slots = {"type": "numbers", "val1": 1, "val2": 2, "word": "two
        c2 = Chunk(name = "c2", slots = {"type": "numbers", "val1": 2, "val2": 3, "word": "thr

        m.add_encounter(c1)
        m.add_encounter(c2)
```

Add some more encounters of these chunks.

```
In [7]: m.time += 15 # Advance the model time by 15 seconds
        m.add_encounter(c2)

        m.time += 20
        m.add_encounter(c1)

        m.time += 5
        m.add_encounter(c2)
```

Let's see what the model looks like now:

```
In [8]: print(m)
```

```
=== Model ===
Time: 40 s
Goal:Chunk goal-chunk
Slots: {'goal': 'count', 'current': 'two'}
Encounters: []
Fan: 0

DM:Chunk c1
Slots: {'type': 'numbers', 'val1': 1, 'val2': 2, 'word': 'two'}
Encounters: [0, 35]
Fan: 0

Chunk numbers
Slots: {}
Encounters: [0, 15, 35, 40]
```

```
Fan: 2

Chunk 1
Slots: {}
Encounters: [0, 35]
Fan: 1

Chunk 2
Slots: {}
Encounters: [0, 15, 35, 40]
Fan: 2

Chunk two
Slots: {}
Encounters: [0, 35]
Fan: 1

Chunk c2
Slots: {'type': 'numbers', 'val1': 2, 'val2': 3, 'word': 'three'}
Encounters: [0, 15, 40]
Fan: 0

Chunk 3
Slots: {}
Encounters: [0, 15, 40]
Fan: 1

Chunk three
Slots: {}
Encounters: [0, 15, 40]
Fan: 1
```

Notice that, even though we've only added two chunks to the model's DM, it contains more chunks. These other chunks are the "singleton" chunks that our own chunks refer to in each of their slots.

1.4 Activation

We can get the activation of a chunk at the current time using the `get_activation()` method.

```
In [9]: print("c1: %f" % m.get_activation(c1))
        print("c2: %f" % m.get_activation(c2))
```

```
c1: 0.498014
c2: -1.026904
```

1.4.1 Spreading activation from goal

The chunk c1 has a slot value in common with the chunk in the goal buffer, which means that there is spreading activation from the goal to this chunk (but not to c2, which does not share any slot values with the goal chunk). We can confirm this by printing the spreading activation on its own:

```
In [10]: print("c1: %f" % m.get_spreading_activation_from_goal(c1))
         print("c2: %f" % m.get_spreading_activation_from_goal(c2))
```

```
c1: 1.000000
c2: 0.000000
```

Right now, the model only implements goal activation spreading. This means that the standard ACT-R spreading equation is simplified a bit. The spreading activation from the goal to chunk i is

$S_i = \sum_j w_j s_{ji}$

where j is the number of sources (i.e., slots) in the goal buffer, w_j is the goal activation (parameter `ga`) divided by j, and s_{ji} is the strength of association from the goal slot j to chunk i, which depends on the maximum spreading association (`mas`) parameter and the fan of the slots in chunk i.

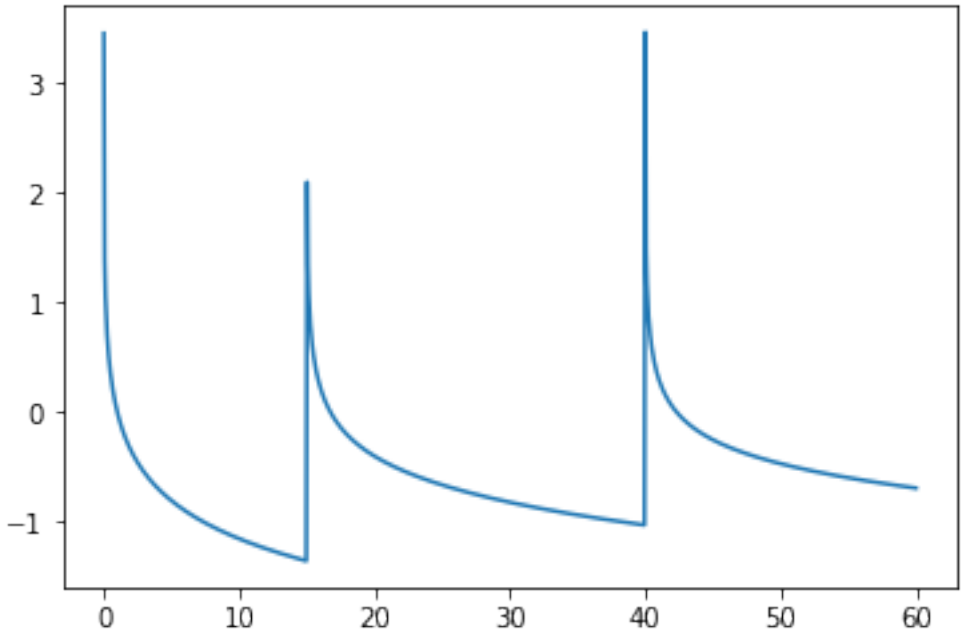
1.4.2 Plotting

Plot the activation of c2 in the first minute:

```
In [11]: x = np.linspace(start = 0, stop = 60, num = 1000)
         bl = []
         for i in x:
             m.time = i + 0.001
             bl.append(m.get_activation(c2))

         plt.plot(x, bl)
```

```
Out[11]: [<matplotlib.lines.Line2D at 0x7fecc5f80eb8>]
```

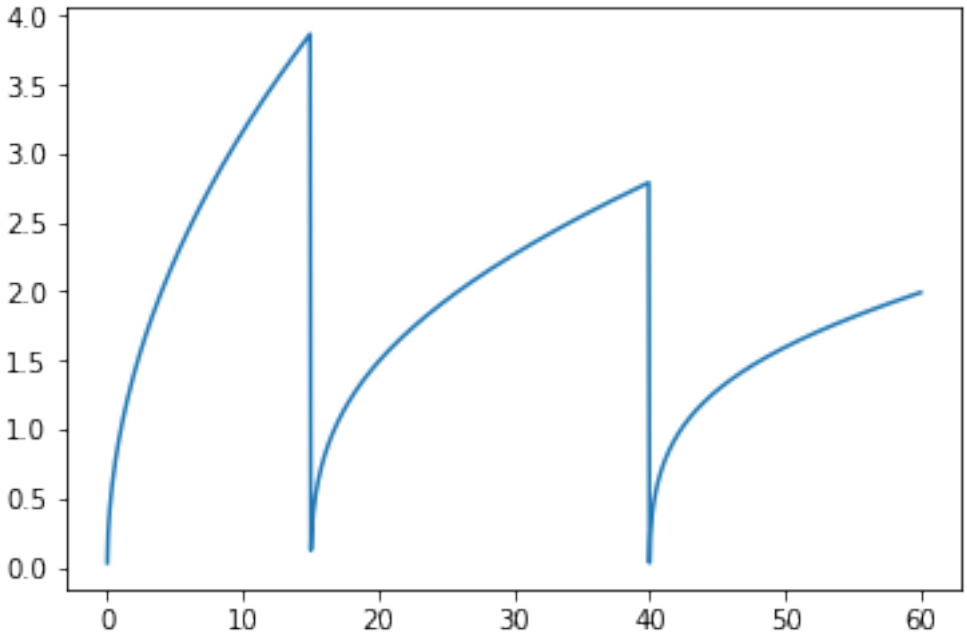


Plot the retrieval latency (directly related to activation) of c2:

```
In [12]: x = np.linspace(start = 0, stop = 60, num = 1000)
         lat = []
         for i in x:
             m.time = i + 0.001
             lat.append(m.get_latency(c2))

         plt.plot(x, lat)

Out[12]: [<matplotlib.lines.Line2D at 0x7fecc5ea20b8>]
```



1.5 Retrieving a chunk from memory

We can retrieve a chunk from memory with a retrieval pattern. We use a Chunk as a representation for that pattern:

```
In [13]: pattern = Chunk(name = "retrieve", slots = {"type": "numbers", "val1" : 1})
```

```
In [14]: chunk, latency = m.retrieve(pattern)
         print(chunk)
         print(latency)
```

```
Chunk c1
Slots: {'type': 'numbers', 'val1': 1, 'val2': 2, 'word': 'two'}
Encounters: [0, 35]
Fan: 0

1.1178539520137154
```

Retrieval failure example:

```
In [15]: pattern = Chunk(name = "test", slots = {"type" : "letters"})
         chunk, latency = m.retrieve(pattern)
         print(chunk)
         print(latency)
```

```
None
2.718281828459045
```

1.6 Blending

Aside from retrieving a sigle chunk, we can also retrieve a *blended trace* (see Taatgen & van Rijn, 2011). This works in much the same way as a normal retrieval, except that we also have to specify the slot for which we want the blended trace.

Blending only works with numerical slot values, so let's make a few new chunks representing some observations of game scores, and add them to the model's DM.

```
In [16]: d1 = Chunk(name = "score1", slots = {"type": "gamescore", "score": 10})
         m.add_encounter(d1)

         m.time += 1

         d2 = Chunk(name = "score2", slots = {"type": "gamescore", "score": 15})
         m.add_encounter(d2)

         m.time += 1

         d3 = Chunk(name = "score3", slots = {"type": "gamescore", "score": 20})
         m.add_encounter(d3)

         m.time += 1
```

We can now retrieve a blended trace of the game score. (Note that we specify a pattern in the same way as before, but that we also tell the model that we want a blended trace of the `score` slot specifically.)

```
In [17]: blend_pattern = Chunk(name = "blended-test", slots = {"type": "gamescore"})

         print(m.retrieve_blended_trace(blend_pattern, "score"))
```

```
18.7707704696836
```

Examples of invalid requests for a blended retrieval:

```
In [18]: # Slot does not exist
         print(m.retrieve_blended_trace(blend_pattern, "non-existstent-slot"))

         # Pattern does not match any chunks in DM
         blend_pattern = Chunk(name = "blended-test", slots = {"type": "letters"})
         print(m.retrieve_blended_trace(blend_pattern, "score"))
```

```
None
None
```