Homework 1 - Travis Hammond - s2880024

November 13, 2020

1 Homework 1 - Addition Model

1.1 Travis Hammond - s2880024

1.1.1 Setup

First, I import Model and Chunk, and create a function that instantiates and initializes a model for me.

[1]:
```
from model import Model
from dmchunk import Chunk


numbers = ['zero', 'one', 'two', 'three', 'four', 'five', 'six']


def init_model():
    # create a model
    m = Model()

    # instantiate the declarative knowledge of how to count to 6
    for num1, num2 in zip(numbers, numbers[1:]):
        fact = Chunk(
            name=f'cf_{num1}-{num2}',
            slots={'isa': 'count-fact', 'num1': num1, 'num2': num2}
        )
        m.add_encounter(fact)

    return m
```

Then, I experiment with code by reimplementing count_from()

[2]:
```
def count_from(m, start, end):
    # formulate the task at hand as a 'goal'
    m.goal = Chunk(
        name='goal',
        slots={'isa': 'count-goal', 'start': start, 'end': end, 'current':␣
 ↪start}
    )
    while not m.goal.slots['current'] == m.goal.slots['end']:
        # formulate a request for the next number after 'current'
        request = Chunk(
            name='request',
            slots={'isa': 'count-fact', 'num1': m.goal.slots['current']}
        )
        # add the time it takes to create the request
        m.time += 0.05

        # retrieve the chunk from declarative memory
        chunk, latency = m.retrieve(request)
        m.add_encounter(chunk)
        # add the time it takes to retrieve the chunk
        m.time += latency

        # add the time it takes to say a number
        m.time += 0.3
        # print the number that was just said and the time elapsed
        print('Alice says:', m.goal.slots['current'])
        print('Time taken:', round(m.time, 2))

        # update current so we can look for the next number
        m.goal.slots['current'] = chunk.slots['num2']

    # add the time it takes to say a number
    m.time += 0.3
    # print the number that was just said and the time elapsed
    print('Alice says:', m.goal.slots['current'])
    print('Time taken:', round(m.time, 2))
```

Let's test the code so far:

[3]:
```
m = init_model()
count_from(m, "two","five")
print(m)
```

```
Alice says: two
Time taken: 0.36
Alice says: three
Time taken: 0.76
Alice says: four
Time taken: 1.15
Alice says: five
Time taken: 1.45

=== Model ===
Time: 1.4526865987160247 s
```

```
Goal:Chunk goal
Slots: {'isa': 'count-goal', 'start': 'two', 'end': 'five', 'current': 'five'}
Encounters: []
Fan: 0

DM:Chunk cf_zero-one
Slots: {'isa': 'count-fact', 'num1': 'zero', 'num2': 'one'}
Encounters: [0]
Fan: 0

Chunk count-fact
Slots: {}
Encounters: [0]
Fan: 6

Chunk zero
Slots: {}
Encounters: [0]
Fan: 1

Chunk one
Slots: {}
Encounters: [0]
Fan: 2

Chunk cf_one-two
Slots: {'isa': 'count-fact', 'num1': 'one', 'num2': 'two'}
Encounters: [0]
Fan: 0

Chunk two
Slots: {}
Encounters: [0]
Fan: 2

Chunk cf_two-three
Slots: {'isa': 'count-fact', 'num1': 'two', 'num2': 'three'}
Encounters: [0, 0.05]
Fan: 0

Chunk three
Slots: {}
Encounters: [0]
Fan: 2

Chunk cf_three-four
Slots: {'isa': 'count-fact', 'num1': 'three', 'num2': 'four'}
Encounters: [0, 0.41170212424176333]
```

```
Fan: 0

Chunk four
Slots: {}
Encounters: [0]
Fan: 2

Chunk cf_four-five
Slots: {'isa': 'count-fact', 'num1': 'four', 'num2': 'five'}
Encounters: [0, 0.8096574946787579]
Fan: 0

Chunk five
Slots: {}
Encounters: [0]
Fan: 2

Chunk cf_five-six
Slots: {'isa': 'count-fact', 'num1': 'five', 'num2': 'six'}
Encounters: [0]
Fan: 0

Chunk six
Slots: {}
Encounters: [0]
Fan: 1
```

1.1.2 Actual code

Then, I implement the add() function. I initially considered not (mentally) incrementing the counter, because I think humans usually use their fingers to keep track of the counter variable in this style of counting (probably due to the fact that the same mental resources are used, making it easy to get mixed up).

In my opinion this could be changed by keeping the formulation of the counter-incrementing request (and the 50ms that takes), and then send that request to the motor module to raise another finger. This happens in parallel to vocalizing, so it does not take any extra time, and it does not interfere with the same mental resource as the sum-incrementing because it relies on muscle memory. Possibly an extra time increase could be added for checking whether the finger count matches the num2 variable.

However this seemed like overkill, and we don't have a motor module available anyway, so I implemented the adding by mentally incrementing both the sum and the counter.

[4]:
```python
def add(m, num1, num2):
    # formulate the task at hand as a 'goal'
    m.goal = Chunk(
        name='goal',
        slots={'isa': 'add-goal', 'sum': num1, 'counter': 'zero'}
    )

    while not m.goal.slots['counter'] == num2:
        # formulate a request for the next number after the current sum
        sum_request = Chunk(
            name='request',
            slots={'isa': 'count-fact', 'num1': m.goal.slots['sum']}
        )
        # add the time it takes to create the request
        m.time += 0.05
        # retrieve the chunk from declarative memory
        sum_chunk, sum_latency = m.retrieve(sum_request)
        m.add_encounter(sum_chunk)
        # add the time it takes to retrieve the chunk
        m.time += sum_latency

        # formulate a request for the next number after the current counter
        cnt_request = Chunk(
            name='request',
            slots={'isa': 'count-fact', 'num1': m.goal.slots['counter']}
        )
        # add the time it takes to create the request
        m.time += 0.05
        # retrieve the chunk from declarative memory
        cnt_chunk, cnt_latency = m.retrieve(cnt_request)
        m.add_encounter(cnt_chunk)
        # add the time it takes to retrieve the chunk
        m.time += cnt_latency

        # add the time it takes to say a number
        m.time += 0.3
        # print the number that was just said and the time elapsed
        print('Alice says:', m.goal.slots['sum'])
        print('Time taken:', round(m.time, 2))

        # update the sum so we know where we are
        m.goal.slots['sum'] = sum_chunk.slots['num2']
        # update the counter so we know when to stop
        m.goal.slots['counter'] = cnt_chunk.slots['num2']

    # add the time it takes to say a number
    m.time += 0.3
    # print the number that was just said and the time elapsed
    print('Alice says:', m.goal.slots['sum'])
    print('Time taken:', round(m.time, 2))
```

Let's run the code and check our results:

```
[5]: m = init_model()
     add(m, 'two', 'four')
     print(m)
```

```
Alice says: two
Time taken: 0.43
Alice says: three
Time taken: 0.91
Alice says: four
Time taken: 1.41
Alice says: five
Time taken: 1.88
Alice says: six
Time taken: 2.18

=== Model ===
Time: 2.18257312992868 s
Goal:Chunk goal
Slots: {'isa': 'add-goal', 'sum': 'six', 'counter': 'four'}
Encounters: []
Fan: 0

DM:Chunk cf_zero-one
Slots: {'isa': 'count-fact', 'num1': 'zero', 'num2': 'one'}
Encounters: [0, 0.11232286120114701]
Fan: 0

Chunk count-fact
Slots: {}
Encounters: [0]
Fan: 6

Chunk zero
Slots: {}
Encounters: [0]
Fan: 1

Chunk one
Slots: {}
Encounters: [0]
Fan: 2

Chunk cf_one-two
Slots: {'isa': 'count-fact', 'num1': 'one', 'num2': 'two'}
Encounters: [0, 0.572005870607302]
Fan: 0
```

```
Chunk two
Slots: {}
Encounters: [0]
Fan: 2

Chunk cf_two-three
Slots: {'isa': 'count-fact', 'num1': 'two', 'num2': 'three'}
Encounters: [0, 0.05, 1.0572936181901262]
Fan: 0

Chunk three
Slots: {}
Encounters: [0]
Fan: 2

Chunk cf_three-four
Slots: {'isa': 'count-fact', 'num1': 'three', 'num2': 'four'}
Encounters: [0, 0.4806424448338038, 1.5627738983019337]
Fan: 0

Chunk four
Slots: {}
Encounters: [0]
Fan: 2

Chunk cf_four-five
Slots: {'isa': 'count-fact', 'num1': 'four', 'num2': 'five'}
Encounters: [0, 0.9605014048320564]
Fan: 0

Chunk five
Slots: {}
Encounters: [0]
Fan: 2

Chunk cf_five-six
Slots: {'isa': 'count-fact', 'num1': 'five', 'num2': 'six'}
Encounters: [0, 1.4570249282264685]
Fan: 0

Chunk six
Slots: {}
Encounters: [0]
Fan: 1
```

Everything looks good.