Final Report

Ishmam Kabir and Troy Witmer

2025-05-01

1 Objective

Our inital Objective was to use openars to predict future data points on a candlestick graph based on historical data. This uses open, high low, and close data points on each candlestick to predict the next few. This project uses stock data because of its historical availability, accuracy. However due to unforeseen circumstances we could not implement all candlestick data such as high, low, and open, instead we settled for close and changed from a candlestick graph to a line graph. Due to errors in the python version of OpenNARS our program was cut short and was not able to be fully implemented.

2 Problems

During our development we ran into two big problems, first was the prediction rate vs the rate we were sending data to NARS, second was the babbling feature was the only data we were receiving due to issues in the source code. These problems appeared to be seperate at first with our minds focusing on figuring out how we could get NARS to execute a sensorimotor command everytime we sent a sentence to it. After hours of failing to come up with a solution we stumbled across the source code for the SensorimotorChannel which contained bypasses, which according to the comments stated an issue with the Reasoner. After much time debugging, we were unable to find a fix for the issue. As a result of the Reasoner being broken all commands sent from the SensorimotorChannel were random due to the babbling feature.

Here is snippits found inside the source code of OpenNARS Python.

```
// cheating
// since something is wrong the reasoner, such that it
    cannot generate sub-goals correctly, I have to cheat.

reactions, ret = self.input_buffer.buffer_cycle(inputs,
    memory)
for each in reactions:
```

```
    self.reactions.push(each, self.reaction_evaluation(each
        , memory))

return ret
// original
// return self.input_buffer.buffer_cycle(inputs, memory)


// cheating
// since something is wrong the reasoner, such that it
    cannot generate sub-goals correctly, I have to cheat.
// this means there are reactions with no corresponding
    goals
if reaction.goal is None:
     return 0.7
```

3 Method and Results

Despite the issues we still wrote most of the code required to complete our Objective.

Sentence Building

In order to communicate with NARS we constructed sentences relating to the price of closing for that iteration. when NARS's prediction was within a buffer value. To begin with we created the relation for when NARS was in the buffers range:

```
msg = "<{SELF} --> [good]>. %1;0.9%"
```

If NARS fell below the true close then we computed the following:

```
msg_1 = "<{up} --> [on]>. %1;0.9%"

f = str(percent_correct(cur_predicted, expected))

msg_2 = "<{SELF} --> [good]>. %" + f + ";0.9%"
```

If NARS was above the true close then we computed the following:

```
msg_1 = "<{down} --> [on]>. %1;0.9%"

f = str(percent_correct(cur_predicted, expected))

msg_2 = "<{SELF} --> [good]>. %" + f + ";0.9%"
```

To calculate an f value that was meaningful for NARS we decided to use a formula for percent correctness bounded between 0 and 1. If the observed value

was far away from the true value then the function would return a low f value. If the observed was close to the true value then f would be closer to 1, telling NARS that it was "close" to what the truth was.

```
def percent_correct(true, observed):
    if true == 0:
        return 1.0 if observed == 0 else 0.0
    error = abs(observed - true) / abs(true)
    return max(0.0, 1 - error)
```

Our ideal scenario would include obtaining at least one sensorimotor command everytime we sent a sentence to NARS, more if its prediction was way off. Doing so would have allowed us to only have to increment the predicted value of NARS by one per command.

Here is a graph depicting our failure, the only movement is from babbling with an augmentation of NARS's predicted value set to 100

All our candlestick data was obtained via Alpaca API.