060223
Seung Suk Josh Lee

SoftStress—A python software for solving/learning stress patterns

Table of Contents

Stress patterns

- From the StressTyp2 database, I was able to identify 106 unique **patterns** by analyzing their prose descriptions.
- Multiple real-life **languages** can have the same pattern (e.g. 'final stress' is the name of a pattern that multiple languages have).
- These patterns are written in prose description in ST2, e.g.
    - 'Initial': In words of all sizes, primary stress falls on the initial syllable.
    - 'Wargamay': In words of all sizes, primary stress falls on the initial syllable if it has secondary stress, else on the peninitial syllable if has secondary stress. In words of all sizes, secondary stress falls on all heavy syllables. In sequences of light syllables, secondary stress falls on the even numbered syllables, counting from the right edge of the sequence. Secondary stress does not fall on a light syllable which is directly preceded by a heavy syllable. Heavy syllables only occur word initially. There are no light monosyllables.
- From these 106 patterns, we (Brandon and I) identified 61 unique patterns, that had Finite State Automaton generators coded in the ST2.
- This means we can generate the strings that will serve as our data:

    - From the above description, “initial” can be represented with the following set of strings
    - [L1 L], [L1 L L], [L1 L L L], [L1 L L L L], [L1 L L L L L], [L1 L L L L L L] (1 for primary, 2 for secondary stress)
- These 61 patterns are currently what we are working with.
- I am working on adding more patterns to this list of patterns, by turning the prose description into string-generating functions. (Stay tuned for this!)

Quantity Sensitivity

- As we saw above, some patterns are QI and some are QS
    - Some are Quantity Super Sensitive—three way weight distinctions, light-heavy-superheavy.
    - But the 61 patterns we are working with are either QI or QS.
- 28 QI and 33 QS
    - You can see them here: SoftStress_results_initial_weight_1

Quantity Insensitive

- For QI patterns: technically, the least we need is strings of light syllables (see example above for the initial pattern), we can’t just have arbitrarily long words, so we are working with words of 2-7 syllables (6 URs).
- Learning results of most of these patterns are reported in the AMP2022 paper of ours. (me, Cerys, Alessa and Joe, [pdf])

Quantity Sensitive

- For QS patterns: we are working with Light and Heavy syllable combinations up to words of 5 syllables, and + 2 light syllable only string for words of 6 and 7 syllables:

Length	Number of URs
2	4
3	8
4	16
5	32
6	1
7	1

- For example, for words of 2 syllables:
    - LL, LH, HL, HH (N=4)

- Notice 6 and 7 syllable-long words don't have heavy-light combinations, and we do the same, following previous work (e.g., Tesar & Smolensky 2000 paper).
- TS did this because
    - 1. If we had heavy-light combinations for 6/7 syll long words, we would have a LOT more tableaux to deal with;
    - 2. We "probably" don't need to have L and H combinations beyond words of 5 to learn Quantity sensitive patterns;
    - 3. Words longer than 5 syllables are extremely rare in human languages.

Representation

- We want to consider two ways of representing the stress patterns.
    - Foot vs. Grid.
- Foot and Grid, from the modeling perspective, differ in the 'structure/size' of the data and the choice of constraints.
- The UR [L L L] has **12** logically possible Surface Representations (col 3, row 3):
    - [L L L1], [L1 L L], [L L1 L],
    - [L2 L L1], [L L2 L1], [L2 L2 L1],
    - [L1 L2 L], [L1 L L2], [L1 L2 L2],
    - [L2 L1 L], [L L1 L2], [L2 L1 L2]

Length	Number of URs (table above)	Number of SRs For each UR	Col2 X Col3	Number of HRs For each UR	Col2 X Col5
2	4	4	16	6	24
3	8	**12**	96	24	192
4	16	32	512	88	1408
5	32	80	2560	300	9600
6	1	192	192	984	984
7	1	448	448	3136	3136
total	62	768	3,824	4538	15,344

- Now it should make sense why if we have H/L combinations for words of 6/7 syllables, the size of the data would blow up, esp. for QS.
- When we don't have the hidden structure (i.e. if we work with Grids), the SRs are the total number of candidates. (768 for QI, 3824 for QS)
- However, if we assume the foot structure, this means we are assuming hidden structure.
- For example, the same Surface Representation [L L1 L] has three Hidden Representations:

    - [(L L1) L], [L (L1 L)], [L (L1) L]
- Adding these up, the total number of candidates to deal with increases to 15,344 for QS!
- To summarize, changing QI to QS increases the data by adding more tableaux (col 3→col4 and col5→col 6), and changing Grid to Foot increases the data by adding more candidates (HRs) (col 3,4→col 5,6)

Foot

- Additional difference between Foot and Grid comes from the constraints and the violations that they assign to the candidates.
- For example, Foot-based constraints can assign violations to the foot structure:
    - FtBin: assign a violation for every foot that is not bimoraic/disyllabic.
        - L(L) gets 1 violation but L(H) gets 0 violation because (H) is bimoraic.
- For our (Pater, Prickett, Lee) projects currently, we are considering three sets of Foot-constraints:
    - Original Tesar and Smolensky
    - Revised Tesar and Smolensky: main stress assigning constraints counts the number of syllables from the edge to the main-stressed syllable instead of the main-stressed foot
    - Revised Tesar and Smolensky + Nonfin Main: addition of a constraint that penalizes the final syllable with the main stress
    - For the full list of constraint definitions, see our paper [pdf link coming soon]

Grid

- On the other hand, Grid constraints target grid marks on three levels of stress:
    - For example the form [L1 L L2] looks as follows: (remember, no hidden structure)
    - 2 X
      1 X X
      0 XXX
- Naturally, the constraints target different things; e.g., whether there is a clash (or a lapse) at the secondary stress level (level 1, the second floor)
    - 2 X
      1 XX
      0 XXXX This is a violation of *Clash
    - 2 X
      1 X   X
      0 XXXX This is a violation of *Lapse
- We have worked with the grid constraints that Gordon proposed in his 2002 paper, and some revisions of those constraints. (me, Cerys, Alessa and Joe, [pdf] for full list of definitions)
- These Gordon constraints (± revisions in AMP2022 paper) can learn/represent all 28 QI patterns in the ST2.
- To account for the QS patterns, I added the Weight-to-Stress Principle constraint:

    - WSP: assign a violation for every heavy syllable without stress (secondary/main)
- Since there is no feet, the above distinction between (L) and (H) by FtBin, does not hold here. For example, LL and LH would have the exact same violations for all constraints, except for WSP.
- So far, adding WSP allowed us to learn 12/33 QS patterns with the Grid-based constraints (13 if we increase the number of epochs to learn)
    - I don't know if these 13 included any of the DTO or DTS patterns.
- Obviously, we would need more constraints to learn all of the QS patterns.

Representability vs. Learnability

- We want to know how to represent a stress pattern with the weighted constraints grammar and we want to know whether a learner can learn any of those grammars.

Weighted Constraint grammar

- What does it mean to represent a pattern with the weighted constraints?
    - For each UR, we have a set of candidates:
        - SRs for Grid, SR-HRs for Foot
    - Each candidate is evaluated by a set of constraints (violations are negative numbers) that each has an associated weight
    - Thus we have a tableau of 'Number of Candidates X Number of Constraints'
    - The harmony is calculated by computing the sumproduct of the violations and the weights of the constraints.
    - Only SR is the observed form, we call this SR 'the winner'.
    - We want to know the weights of the constraints such that for each UR tableau, the harmony of the winner is the biggest out of the candidates that the winner is competing with.
        - Biggest means closer to 0
        - (e.g., if an SR is perfect in a tableau, and therefore is a winner, it would get 0 violation from the constraints, therefore it will have Harmony of 0)
        - (There are infinite number of weight vectors that can satisfy this because the weights can be scaled up infinitely)
    - If such weight exists, then we say the pattern is **representable** by the current grammar.
- We can either **solve** or **learn** a grammar (set of weights), I'll explain them briefly first:
- **Solving** a grammar is defined here as analytically computing the weight vector via what's called Linear Programming such that the sum of weights is minimized while satisfying the necessary condition of making the winner have the largest harmony
- **Learning** a grammar means we want to find the set of weights in a way that we think is cognitively plausible, in the sense that the weights are updated incrementally, based on the error that the learner makes, in each iteration (epoch).
- It is worth noting that learnability guarantees solvability
    - If a grammar is learnable, then it must be solvable.

- It is crucial to note that for Grid, solvability guarantees learnability too.
  - If a grammar is solvable, then it must be learnable.
- However, for Foot, solvability does not guarantee learnability, because of hidden structure (if there is a local minimum).
  - If a grammar is solvable, it does not always mean it's learnable.

Solving (Representability)

- We can **solve** a grammar by solving a series of inequality equations.
- This is explained in detail in Potts et al. 2010 paper and I repeat their example below:

(10)

Input_1	C_1	C_2
a. Winner_1	0	−2
b. Loser_1	−6	0

Input_2	C_1	C_2
a. Winner_2	−1	0
b. Loser_2	0	−1

- This language can't be represented with OT because of the inconsistency
- It is representable with HG because even if W(C2) > W(C1), the harmony of loser1 should still be smaller (further away from zero) than the harmony of winner1, because it has more violations of C1.
- This can be expressed with a series of inequalities:

(11) a. $(0 \cdot W(C_1)) + (-2 \cdot W(C_2)) > (-6 \cdot W(C_1)) + (0 \cdot W(C_2))$
b. $(-1 \cdot W(C_1)) + (0 \cdot W(C_2)) > (0 \cdot W(C_1)) + (-1 \cdot W(C_2))$

For the numerical optimisations to follow, we make extensive use of the following notation.

(12) a. $0w_1 + -2w_2 > -6w_1 + 0w_2 \Rightarrow 6w_1 + -2w_2 > 0$
b. $-1w_1 + 0w_2 > 0w_1 + -1w_2 \Rightarrow -1w_1 + 1w_2 > 0$

- Then, a "linear programming problem" (LP) can be set up as below

$$\begin{array}{ll} \text{minimise} & 1w_1 + 1w_2 \\ \text{subject to} & 6w_1 - 2w_2 \geq 1 \\ & -1w_1 + 1w_2 \geq 1 \\ & 1w_1 \geq 1 \\ & 1w_2 \geq 1 \end{array}$$

- Instead of setting it as ≥0, we set up as ≥alpha (1, in this example), because we don't want things to tie
  - In this particular example, there's also a minimal weight constraint (≥1), in our projects, our minimal weight is 0 (no neg weights)
- Summary:

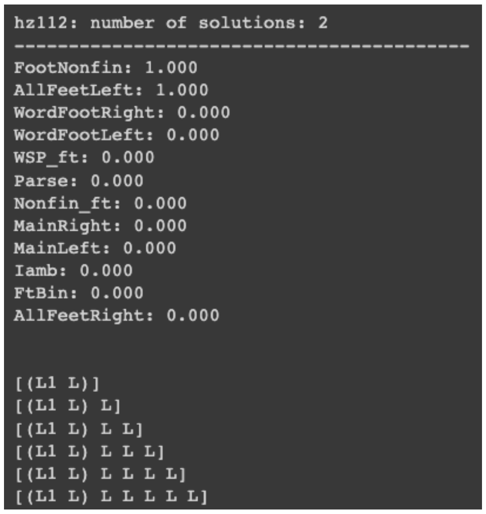
- Given a list of Tableaux T (length n, e.g., n=62 for our QS case)
- In each T, we have candidates Cand (length i)
- Each candidate Cand_i can be expressed as a sumproduct of a Violation vector for a list of constraints and a Weight vector of those constraints
- One candidate is the winner (let's say the first candidate, for example), and the others are losers
- Then we have the following list of inequality relations:
    - (i-1 inequalities for each tableau, because winner_id vs. all the other ids)
      $T_1\text{Cand}_{\text{winner}} > T_1\text{Cand}_2 + \text{alpha}$
      $T_1\text{Cand}_{\text{winner}} > T_1\text{Cand}_3 + \text{alpha}$
      …
      $T_1\text{Cand}_{\text{winner}} > T_1\text{Cand}_i + \text{alpha}$

  Moving on to the next tableau...
      $T_2\text{Cand}_{\text{winner}} > T_2\text{Cand}_i + \text{alpha}$
      …
      $T_n\text{Cand}_{\text{winner}} > T_n\text{Cand}_i + \text{alpha}$
- Each constraint weight ≥ 0 (no negative weight)
- Objective: minimize the sum of weights to satisfy all the inequalities above

● This Linear Programming method is also implemented with Javascript (needs confirmation) in OT-help, proposed in Potts et al. 2010.
  - However, OT-help seems to fail with larger problems (e.g., our problem with 62 tableaux and over 10,000 HR candidates)

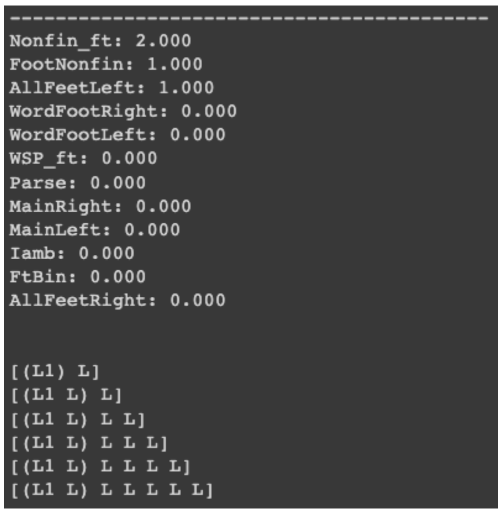
● Grid: only one solution exists because only one candidate can be the winner

● Foot: potentially, multiple solutions exist because from the observed data, we know an SR is the winner, but that SR has multiple consistent HRs.
  - For example, you will see in the demo (that appears later in this manual) that the initial stress (QI) pattern can have two solutions with the original TS constraints:



```
hz112: number of solutions: 2
----------------------------------------
FootNonfin: 1.000
AllFeetLeft: 1.000
WordFootRight: 0.000
WordFootLeft: 0.000
WSP_ft: 0.000
Parse: 0.000
Nonfin_ft: 0.000
MainRight: 0.000
MainLeft: 0.000
Iamb: 0.000
FtBin: 0.000
AllFeetRight: 0.000


[(L1 L)]
[(L1 L) L]
[(L1 L) L L]
[(L1 L) L L L]
[(L1 L) L L L L]
[(L1 L) L L L L L]
```



```
----------------------------------------
Nonfin_ft: 2.000
FootNonfin: 1.000
AllFeetLeft: 1.000
WordFootRight: 0.000
WordFootLeft: 0.000
WSP_ft: 0.000
Parse: 0.000
MainRight: 0.000
MainLeft: 0.000
Iamb: 0.000
FtBin: 0.000
AllFeetRight: 0.000


[(L1) L]
[(L1 L) L]
[(L1 L) L L]
[(L1 L) L L L]
[(L1 L) L L L L]
[(L1 L) L L L L L]
```

Learning (Learnability)

- As for the learner, we have been using a MaxEnt learner that can handle hidden structure. (alternative appraoch: Gaja's EDL)
- MaxEnt probabilities are computed with the Harmony scores, by first exponentiating them and normalizing them (see e.g., Hayes & Wilson 2008). MaxEnt
- Each learning update is determined by how different the probabilities computed by the current grammar are from the actual probabilities.
- The default values we used are 1000 epochs (1000 updates) and update step (aka learning rate) of 4, and we decided that the pattern is learned, if for every UR, the correct observed SR form gets the probability of 0.9
  (Maybe there's a better demonstration of how this works out there, it would be great if I could have a link to one of those demos here)
- Grid: when there's no hidden structure, this means the candidate that gets a probability of 0.9, will be the winner SR candidate.
- Foot: if an SR wins, then potentially, multiple consistent HRs can jointly have .90 prob.
  - E.g., if in the tableau of [L L L], the winner SR is [L L1 L], then potentially, it could be that the learner assigns a .8 probability to [(L L1) L]
    and .1 probability to [L (L1 L)], making the .9 probability jointly.
  - We've definitely seen this happening in our learning simulations.

Demo–SoftStress

- Now let's see a demonstration of the code I wrote to solve/learn QI/QS patterns in the ST2 database, with Grid/Foot constraints
- Summary:
  - final_weights, learned_when = learn_language(filename, QI_or_QS, Foot_or_Grid, Constraint set)
  - print_result_pretty(filename, QI_or_QS, Foot_or_Grid, Constraint set, final_weights, learned_when)

  - solutions = solve_language(filename, QI_or_QS, Foot_or_Grid, Constraint set)
  - print_solutions_pretty(filename, QI_or_QS, Foot_or_Grid, Constraint set, solutions)

Summary of results so far

- Best results

QI (Grid > Foot)
Foot
RevisedTS + NonFinMain with 6 URs

QI	Learned (22)	Not learned (6)
Solved	111, 112, 113, 115, 117,	212

	118, 119, 128, 129, 131, 136, 144, 145, 150, 157, 158, 170, 179, 208, 211, 213, 215	
Not solved	■	133, 134, 169, 171, 191

Grid
Original/Revised Gordon with 6URs

QI	Learned (all 28)	Not learned
Solved	111, 112, 113, 115, 117, 118, 119, 128, 129, 131, 133, 134, 136, 144, 145, 150, 157, 158, 169, 170, 171, 179, 191, 208, 211, 212, 213, 215	
Not solved	■	

QS (Foot > Grid, slightly)
Foot
RevisedTS + NonFinMain

QS	Learned (15)	Not learned (18)
Solved	116, 132, 148, 153, 156, 159, 162, 167, 177, 178, 192, 199, 205, 214, 218	Not checked yet
Not solved	■	114, 127, 139, 146, 149, 165, 172, 176, 182, 184, 194, 196, 200, 203, 204, 209, 210, 219

Grid
Original Gordon + WSP

QS	Learned (12)	Not learned (21)
Solved	116, 132, 148, 156, 159, 162, 165, 177, 200, 205, 214, 218	192
Not solved	■	114, 127, 139, 146, 149, 153, 167, 172, 176, 178, 182, 184, 194, 196, 199, 203, 204, 209, 210, 219

Future directions

Research topics/questions

- There are patterns that are representable with a set of constraints, and yet not learnable with the same constraints. What kind of local minima cause this?
    - Are there any characteristics among these patterns?
- When a pattern is learned, it learns one of potentially many solutions. What solution is selected and why?
- **What more constraints do we need to learn the patterns that are not currently solvable? (feet vs. grid)**
    - **Are there any characteristics among these unsolvable/unlearnable patterns?**
- For QI languages, what's the effect of learning with 62 URs vs. 6 URs?
    - If only considering 6 URs helps learner learn the pattern, from the input data, how does a learner know whether they're learning a QI/QS pattern?
- TS and our previous work have used L and H combo upto words of 5 sylls, but not for 6 or 7. What happens if we include L and H combo for 6 or 7, for those currently unlearnable patterns?
- Why are some patterns easier with Grid than with Feet, and vice versa?

Planned updates

- More patterns from ST2
- Adding the st2 database table into the software, so that it's easy to search the pattern with the strings, or search the strings with the pattern name, or a particular language name, etc. (almost ready)
    - Using the ST2 database table directly as input data for solver/learner
- More/better result reporting/analyzing functions? Need more user feedback