Class 16 (Week 8, T)
Learning Models III: numerical learning algorithms for constraint grammars

To do
- ☐ Read **Kirby (2013)** for Thursday (Nov. 19)
  - presenters, if you e-mail me your handout as a PDF by noon Thurs., I can print
- ☐ Prepare at least one **question or point for discussion** on the reading
- ☐ Computing **homework** using OTSoft is due Tuesday (Nov. 24). Turn in write-up on paper; I will enable file upload on CCLE for output files.

Overview: We know how a grammar works once we have a ranking or set of weights, but how does the learner arrive at that ranking/set of weights? Last week we looked at ranking (no numbers) and hidden structure. Now we look at learning numerical weights.

1. MaxEnt review

- The learner's job is to maximize
  - *log probability that grammar assigns to observed data – penalty for non-default weights*
  - $= \sum_{j=1}^{m} \ln(p(datum_j \mid grammar)) \; - \; \sum_{i=1}^{n} weightPenaltyForConstrain_i$
  - $= \sum_{j=1}^{m} \ln\left(\frac{e^{-sumOfWeightedConstrainViolations_j}}{total\_e^{-sumWghtedConstrViol\_ForAllCandsInTableau_j}} \right) - \sum_{i=1}^{n} \frac{(actualWeight_i - defaultWeight_i)^2}{2willingnessToChange_i^2}$
  - $\sum_{j=1}^{m} \ln\left(\frac{e^{-\sum_{i=1}^{n} w_i C_i(datum_j)}}{Z_j} \right) - \sum_{i=1}^{n} \frac{(w_i - \mu_i)^2}{2\sigma_i^2}$
  - $= \sum_{j=1}^{m} (-\sum_{i=1}^{n} w_i C_i(datum_j) - \ln(Z_j)) \; - \sum_{i=1}^{n} \frac{(w_i - \mu_i)^2}{2\sigma_i^2}$

  I hope I got all the negative signs right!

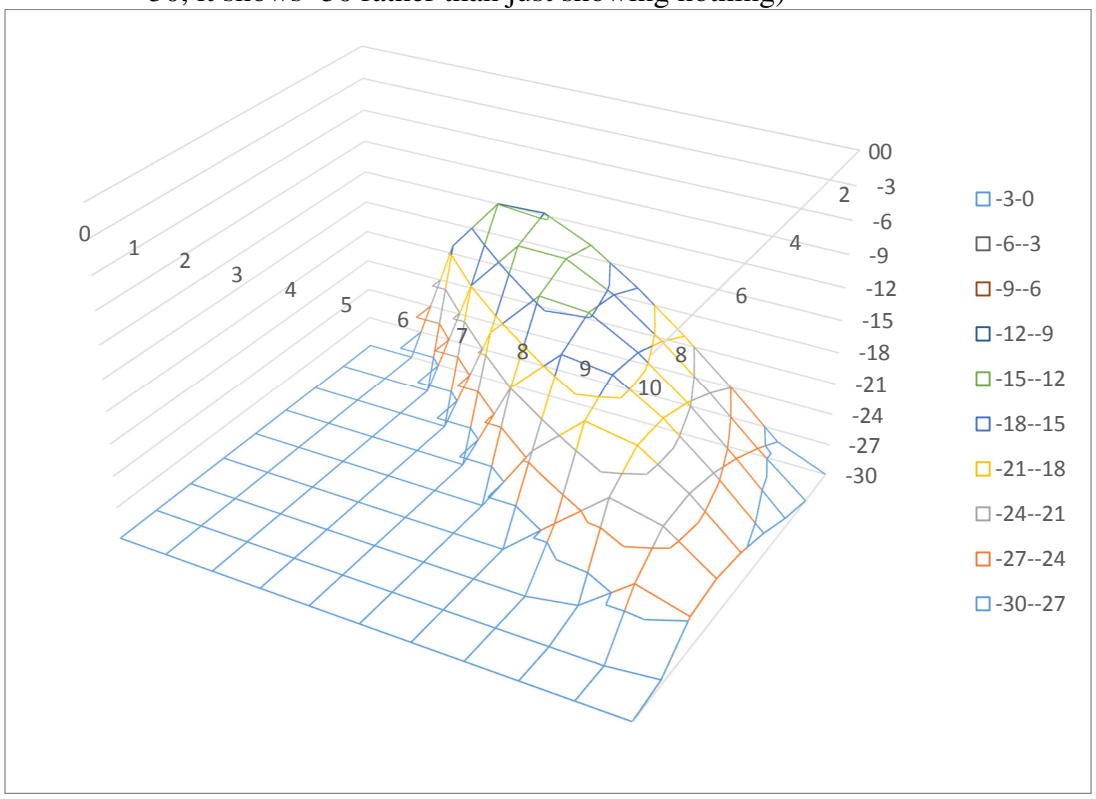
2. But how does the computer find the best weights?

(The good news: this part is really not relevant for us—the computer does it all.)

- Some form of *gradient ascent/descent*
- Simple Excel example: two constraints (I'll show this on the screen)

	sigmas:	2	2
	mus:	0	0
/rad/	observed prob.	Ident(voice)	*FinalVoicedObstruent
[rad]	2	0	1
[rat]	98	1	0

- A plot of the objective function, for various values of the two weights:
  - (Excel quirk: because I limited the z-axis to -30 through 0, for all areas that fall below -30, it shows -30 rather than just showing nothing)



  - The learner doesn't have access to this full landscape
  - But, it can start somewhere, and figure out the local **gradient** (slope)
    - How? A whole bunch of linear algebra.
    - Some methods move in just one dimension at a time (i.e., change one weight), while others can move in any direction.
    - See Shewchuk (1994) for a tutorial on one method (still difficult for me, despite title)
  - Then, the learner just has to take a small step uphill, check the gradient again, and repeat.
  - Terminology: people talk about *gradient descent* when they want to minimize something, like error, and *gradient ascent* when they want to maximize something, like probability.
- The important thing is that in MaxEnt, we are guaranteed that the surface is **convex**
  - As long as we keep going uphill, we'll get to the peak (global optimum).
  - We don't have to worry about getting stuck at a local optimum.

3. Noisy Harmonic Grammar review

- Draw a Harmonic Grammar tableau for the schematic example above. Pick weights that make [rat] the winner.

- How do you make Harmonic Grammar "noisy"?

4. A way to find HG weights: the Gradual Learning Algorithm

- The Gradual Learning Algorithm is *error-driven*.
  - What makes a learning algorithm error-driven?

- How it works for Noisy Harmonic Grammar
  - Hear an utterance
    - Will usually be [rat], but sometimes will be [rad]
  - Use the current grammar to choose a candidate (flip coin to break tie)
    - With starting weights as shown, what will happen?

weight	0	0
/rad/	IDENT(voice)	*FINALVOICEDOBSTRUENT
[rad] (2%)		*
[rat] (98%)	*	

  - If the chosen candidate doesn't match what you heard...
    - slightly increase the weight of any constraints that prefer the winner
    - slightly decrease the weight of any constraints that prefer the loser
    - Compare and contrast with Error-Driven Constraint Demotion

  - Repeat.

- There's no point when the learner knows it's done
  - If adult data is variable, then the grammar can never completely stop making errors
  - A typical strategy is to keep lowering the increment/decrement value until it's zero.
  - Another possibility is just to stop running the learner after a certain number of iterations.
    - You could also do both—lower the increment/decrement, but stop before it hits zero.

5. Stochastic OT (Boersma 1998, Boersma & Hayes 2001 for a more tutorial presentation)

- We're now in a good position to go over another model of variation in constraint grammar, which you read about in Jarosz (submitted)
- It works a lot like Noisy HG, in that...
  - Each constraint has a number
    - called "ranking value" rather than "weight"
  - Each time you want to talk, you add a little noise, drawn from a normal distribution

ranking value	37.0	40.0
noise on this occasion	0.7	-1.2
ranking value + noise ("selection point")	37.7	38.8
/rad/	IDENT(voice)	*FINALVOICEDOBSTRUENT
[rad]		*
[rat]	*	

- The difference is that instead of adding up harmony to find the best candidate...
  - You just rank the constraints according to their selection points
  - And apply strict-ranking OT in the usual way
- Also, it's fine for ranking values to be negative, since the absolute numbers don't matter, only the relative ranking that results.

- As with Noisy HG, the easiest way to get candidate probabilities from ranking values is to simulate them (e.g., run the grammar 10,000 times)
  - But you can also do numerical integration
- As with Noisy HG, the most popular learner is the Gradual Learning Algorithm

6. Problems with the Gradual Learning Algorithm

- Unlike MaxEnt, it's not guaranteed to work.
- The GLA can fail to converge, e.g., demoting one constraint ever more
- If the GLA does converge, we have no guarantee that the ranking values it finds are the "best"
  - e.g., these are the ranking values that produce the lowest error rate, or that maximize the probability of the observed data
- If you want to know more: Boersma & Pater 2013; Pater 2008; Magri 2012

7. Partial Ordering OT

- A Stochastic OT grammar defines a probability distribution over strict-ranking OT grammars
  - E.g., 50% A>>B, 50% B>>A
  - or 10% A>>B, 90% B>> A
    - How would the difference between these two grammars be represented?

- Not just any distribution, though—it has to be capturable with ranking values
  - impossible: 80% A>>B>>C, 20% C>>A>>B, 0% for the rest
    - Why impossible?

- Anttila (1997) proposes an even more restricted theory of probability distributions over strict rankings
  - Define a **strict partial order** over constraints
    - Anyone who's taken Math Ling: remind us what the crucial properties of a strict partial order are.

    - Draw any strict partial order of 5 constraints, A, B, C, D, E

  - Enumerate all the total orderings consistent with that partial order
  - Each of those total orderings is equally probable
    - I.e., if there are *n* such total orderings, then each one's probability is _____.

- As far as I know no learning algorithm has been proposed for this theory.
  - But you can sort of fake it with Stochastic OT and the Gradual Learning Algorithm:
    - Set the learning increment/decrement to be huge for the whole learning period, e.g., 20
    - The result is that the constraints effectively sort themselves into strata
    - This can't capture a non-stratified partial order, though

8. Grammar models roundup

	model name	*typical learning algorithms*	*easiest way to get candidate probabilities*
probability distribution over strict rankings	Classic OT (trivially)	Error-Driven Constraint Demotion Recursive Constraint Demotion	inspect tableau
	Stochastic OT	Gradual Learning Algorithm	simulate
	Partial Ordering OT	?	enumerate linear rankings
	Pairwise Ranking Grammar	Expectation Maximization	simulate
constraint weighting	Harmonic Grammar	Gradual Learning Algorithm	simulate
	MaxEnt	Gradient descent/ascent	calculate directly from grammar

9. Learner roundup

- Some of our algorithms are guaranteed to work (at least on their own terms), others not

- We've seen that some of our learners were **batch** and others **on-line**—discuss the difference, pros and cons

10. Back to the question of smoothing/regularization (and other priors)

- Recall: the purpose of smoothing is _____

- Smoothing will always result in a _____ fit to the training data.
- So how do we *know* whether we're over- or under-fitting?
- Ideally, we would be able to test different grammars, say against wug-test data.
  - The one that best predicts the wug test data (after training on existing data) is the one that was the right fit.
- This is similar to the machine-learning perspective:
  - The right fit is the one that does the best job of predicting new data
- But what if we don't have new data?

11. Cross-validation

- Randomly divide your data into two sets.
- One set is the training data
  - Fit various models to it

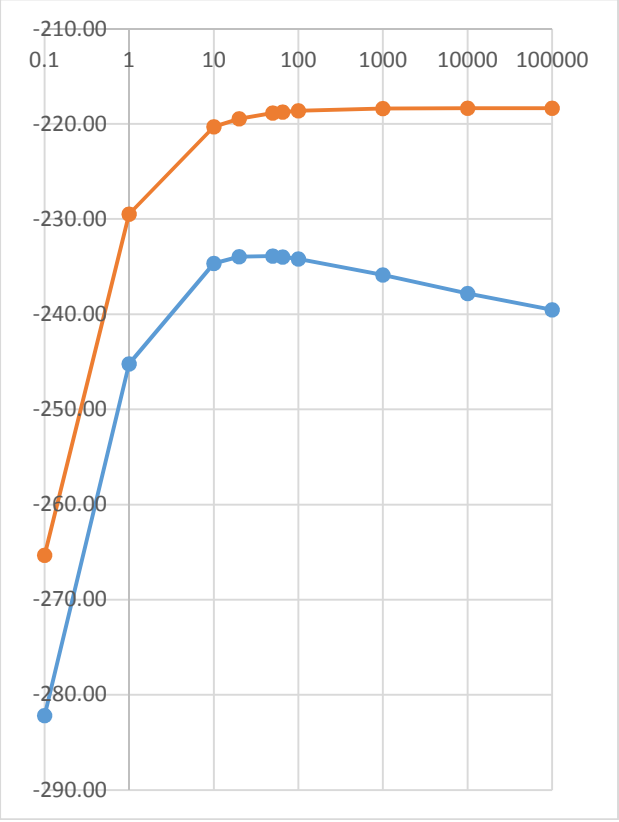
- The other[1] is the cross-validation set
  - See how well each model does at predicting these "new" data

(Another common approach: divide the data in tenths, and do the whole procedure 10 times, each time using a different tenth as the cross-validation set.)

I tried it on Shona

- Randomly selected 20% of tokens to hold back[2]
- Fitted a series of a series of MaxEnt grammars to the other 80% of the data
  - All have the same constraints
  - For simplicity of cross-validation, just two kinds
    - DEP-*X* (5 of them, one for each vowel): always giant σ^2 of 100,000
    - DEP-*X*/C__T (5), DEP-*X*/NearestVIs*Y* (25), DEP-*X*/PrecedingCIs*Y* (20): σ varies
- Test each grammar on the 20%
- Check the fit (log likelihood—the first term of the objective function)
  - log likelihood of test data under the grammar
  - log likelihood of training data under the grammar (divided by 4 for easier graphical comparison)
  - Discuss the results:

σ^2 for complex constraints	log likelihood of test data	log likelihood of training data	(log likelihood of training data)/4
100000	-239.54	**-873.41**	-218.35
10000	-237.81	-873.43	-218.36
1000	-235.88	-873.57	-218.39
100	-234.19	-874.55	-218.64
65	-233.98	-875.05	-218.76
50	**-233.89**	-875.46	-218.86
20	-233.94	-877.80	-219.45
10	-234.66	-881.21	-220.30
1	-245.23	-917.93	-229.48
0.1	-282.18	-1061.32	-265.33



[1] Usually there's yet a third set, the testing data, used for getting an estimate of model performance on new data, once the cross-validation set has been used to tweak the parameters. You can think about the following: why can't you just use the cross-validation set for this purpose?

[2] Used the rbinom() function in R.

12. Coming up

- Phonologization: How do learners end up turning phonetics into phonology?

References

Anttila, Arto. 1997. Deriving variation from grammar. In Frans Hinskens, Roeland van Hout & W. Leo Wetzels (eds.), *Variation, Change, and Phonological Theory*, 35–68. Amsterdam: John Benjamins.

Boersma, Paul. 1998. *Functional Phonology: Formalizing the Interaction Between Articulatory and Perceptual Drives*. The Hague: Holland Academic Graphics.

Boersma, Paul & Bruce Hayes. 2001. Empirical tests of the gradual learning algorithm. *Linguistic Inquiry* 32. 45–86.

Boersma, Paul & Joe Pater. 2013. Convergence properties of a Gradual Learning Algorithm for Harmonic Grammar. Manuscript. University of Amsterdam and University of Massachusetts, Amherst, ms.

Hayes, Bruce. 2004. Phonological acquisition in Optimality Theory: The early stages. In René Kager, Joe Pater & Wim Zonneveld (eds.), *Constraints in Phonological Acquisition*. Cambridge: Cambridge University Press.

Jarosz, Gaja. submitted. Expectation driven learning of phonology. UMass, ms.

Kirby, James. 2013. The role of probabilistic enhancement in phonologization. In Alan C. L. Yu (ed.), *Origins of sound change*, 228–246. Oxford: Oxford University Press.

Magri, Giorgio. 2012. Convergence of error-driven ranking algorithms. *Phonology* 29(02). 213–269.

Pater, Joe. 2008. Gradual Learning and Convergence. *Linguistic Inquiry*.

Prince, Alan & Bruce Tesar. 2004. Learning phonotactic distributions. In René Kager, Joe Pater & Wim Zonneveld (eds.), *Constraints in Phonological Acquisition*, 245–291. Cambridge: Cambridge University Press.

Shewchuk, Jonathan Richard. 1994. An introduction to the Conjugate Gradient Method without the agonizing pain. Manuscript. Carnegie Mellon University, ms.

Tesar, Bruce. 1999. Robust interpretive parsing in metrical stress theory. In Kimary N Shahin, Susan J Blake & Eun-Sook Kim (eds.), *The Proceedings of the West Coast Conference on Formal Linguistics 17*. Stanford, CA: CSLI Publications.

Tesar, Bruce & Paul Smolensky. 2000. *Learnability in Optimality Theory*. Cambridge, Mass.: MIT Press.

Tessier, Anne-Michelle & Karen Jesney. 2014. Learning in Harmonic Serialism and the necessity of a richer base. *Phonology* 31(01). 155–178.

Zuraw, Kie, Kristine Mak Yu & Robyn Orfitelli. 2014. Word-level prosody in Samoan. *Phonology* 31(2). 271–327.