ERROR-DRIVEN LEARNING IN OT AND HG: A COMPARISON

GIORGIO MAGRI

[to appear with minor modifications in *Phonology*]

Abstract. The OT error-driven learner is known to admit guarantees of efficiency, stochastic tolerance and noise robustness which hold independently of any substantive assumptions on the constraints. This paper shows that the HG learner instead does not admit such constraint-independent guarantees. The HG theory of error-driven learning thus needs to be substantially restricted to specific constraint sets.

1. Introduction

An *error-driven* learner entertains a current grammar which is slightly updated whenever it makes an error on the current piece of data. The *computational theory* of error-driven learning provides guarantees for four core requirements: *convergence* (is the number of errors made by the learner finite?), *efficiency* (is the number of errors furthermore small?), *stochastic tolerance* (does the number of errors remain small for the stochastic implementation?), and *noise robustness* (does it remain small in the presence of noisy data?).[1] Within *constraint-based phonology*, these computational guarantees can be derived from two perspectives (Tesar and Smolensky 2000, section 1.2): the *constraint-independent* perspective derives computational guarantees from just the formal mode of constraint interaction; the *constraint-dependent* perspective instead resorts to substantive assumptions on the phonological constraints. This paper compares error-driven learning within two frameworks for constraint-based phonology: *Harmonic Grammar* (HG; Legendre, Miyata, and Smolensky 1998b,a; Smolensky and Legendre 2006) and *Optimality Theory* (OT; Prince and Smolensky 2004). The OT learner has been endowed with computational guarantees for efficiency, stochastic tolerance, and noise robustness which hold independently of any substantive assumptions on the constraints. This paper shows that the HG learner instead does not admit such constraint-independent guarantees. Any guarantees of efficiency, stochastic tolerance, and noise robustness in HG thus need to be restricted to specific constraint sets.

The paper is organized as follows. Section 2 outlines the logical structure of the computational theory of error-driven learning in constraint-based phonology. Section 3 introduces the OT and HG implementations of error-driven learning. Section 4 compares the two implementations from the perspective of efficiency. It presents a counterexample where the number of errors made by the HG learner grows so fast with the number of constraints that the learner makes over five million errors to weight just ten constraints. Despite the fact that the corresponding HG and OT typologies coincide, the OT learner instead makes less than fifty errors, as expected because of its constraint-independent efficiency guarantees (Tesar and Smolensky 1998; Magri 2012b). Section 5 extends the comparison to the stochastic implementation of error-driven learning. It shows that the number of *additional* errors made by the HG stochastic learner on top of the already astronomical number of errors made by the deterministic learner grows so fast with the number of constraints that over one million additional errors are made in the counterexample with just ten constraints. The OT stochastic learner instead makes only around 150 additional errors, as expected because of its constraint-independent guarantees of stochastic tolerance (Magri 2015c). Finally, section

Date: January 16, 2016.

Parts of this paper have been presented at the 21st Manchester Phonology Meeting in 2013 and at the 11th Old World Conference in Phonology in 2014. I wish to thank Paul Boersma and Joe Pater for useful discussion. Three anonymous reviewers and the associate editor of the journal also provided me with detailed and valuable suggestions. The research reported in this paper has been supported by a grant from the Fyssen Research Foundation as well as by a Marie Curie Intra European Fellowship within the 7th European Community Framework Programme.

[1]These four requirements are singled out here because they have to do with the number of errors and thus pertain *specifically* to the theory of error-driven learning. Of course, there are a number of additional issues (such as restrictiveness, coping with hidden structure, inferring underlying representations, etcetera) which are central to the computational theory of language learning, independently of whether the model adopted is error-driven or not.

6 extends the comparison to the noisy learning setting. It shows that the number of additional errors made by the HG learner because of noisy training data grows so fast that a single small update by a noisy piece of data (which changes at most three of the ten weights by at most 2) requires over 60,000 additional updates to recover. The OT learner instead makes only nine additional errors, as expected because of its constraint-independent guarantees of noise robustness (Magri 2015c). Section 7 concludes with a discussion of the implications of these results for phonological theory and language acquisition.

This paper contributes to the current debate between the OT and HG implementations of constraint-based phonology (Pater 2009) from the perspective of (error-driven) learnability. Even if HG turns out in the end to be better motivated than OT from a typological perspective, the counterexamples discussed in this paper and their detailed analyses still matter, as they focus the HG research agenda on the learnability implications of substantive constraint restrictions.

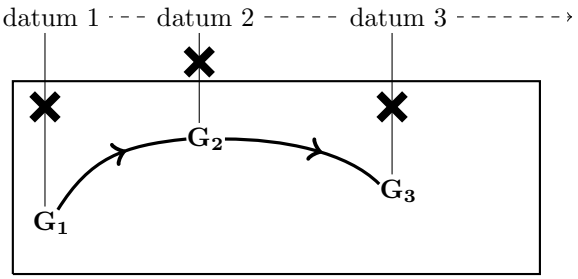
2. The computational theory of error-driven learning in constraint-based phonology

This section outlines the logical structure of the computational theory of error-driven learning in constraint-based phonology, building towards the synopsis in (6).

2.1. **Convergence and efficiency**

Assume that the learner is provided with the space of grammars G_1, G_2, etcetera. The learner maintains a current grammar, which represents its current hypothesis on the target adult grammar. Data come in a stream. Whenever the current grammar (say G_1) makes an error on the current piece of data (say, datum 1), it is updated to a slightly different one (say, G_2). The current piece of data is then discarded and the learner waits for the next piece of data to evaluate the performance of the updated grammar.

(1)



This learning scheme is called *error-driven* because the learning dynamics is driven by the errors made on the incoming stream of data. This scheme has been thoroughly investigated in the machine learning literature (under the heading of *online learning*; for a review, see Kivinen 2003; Cesa-Bianchi and Lugosi 2006, chapters 11, 12; and Mohri et al. 2012, ch. 7). Within the language acquisition literature, this learning scheme has been endorsed at least since Wexler and Culicover (1980) for two reasons. First, an error-driven learner describes a sequence of grammars in the typological space which provides a tool to model child acquisition paths. Second, an error-driven learner does not keep track of previously seen data: the current piece of data is compared with the current grammar and then discarded.[2] It is thus perfectly suited to model the early stages of language acquisition prior to the development of the native language lexicon, such as the early acquisition of phonotactics (Hayes 2004).[3]

[2] The term *error-driven* is used here to refer to a learner with the following *two* properties (see also Tesar 2013, section 5.4.1). First, it maintains a current grammar which is only updated when it makes an error on the current piece of data. Second, it maintains nothing else besides the current grammar. Thus in particular, it does not store any of the data seen at previous iterations. And the update of the current grammar is completely determined by the current piece of data alone, while previously seen data play no role in the definition of the update. This second property is independent of the first property. For instance, Tesar's (2004) *Multi-Recursive Constraint Demotion* (MRCD) is a learner which maintains a current OT grammar which is only updated when it makes an error on the current piece of data, as required by the first property. Yet, MRCD does not comply with the second property, because it also maintains a selection of previously seen data (called the *support*), which partake in the update of the current grammar.

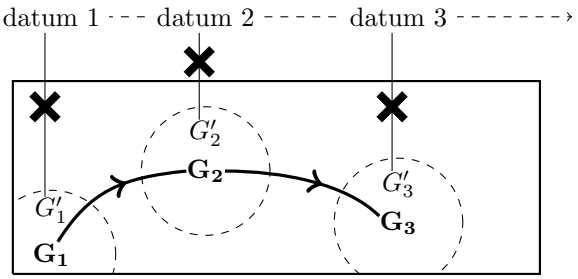
[3] Admittedly, non-error-driven models of the early stage of the acquisition of phonotactics (say, models such as those proposed by Prince and Tesar 2004 and Hayes 2004 within OT) do not really require a stored "lexicon". Rather, they only require a "proto-lexicon" which consists of bare phonological forms, with no morphological decomposition and no corresponding meanings. It is conceivable that the child might be able to assemble this proto-lexicon before any morphological and semantic learning has happened at this early developmental stage. Yet, the point sticks that the burden of proof lies with the proponents of non-error-driven learning: it is up to them to justify the plausibility of their modeling assumptions by providing psycholinguistic evidence that such proto-lexica are indeed stored and already available in the very first months of life. Error-driven learning thus does have a leg up, in the sense that it is less demanding from a psycholinguistic perspective. For instance, Gibson and Wexler (1994, p. 410) underline as one of the virtues of error-driven learning the fact that "it requires no memory of either earlier data or previous parameter settings, so as to limit reliance on the child's memory."

The most basic question of the computational theory of error-driven learning concerns *convergence*: is it possible to guarantee that the learner only makes a finite number of errors? Convergence is crucial because it means that the learner eventually settles on a final grammar which will never be further updated and thus counts as the grammar *learned* by the algorithm. Of course, there is little difference between the number of errors being infinite and it being finite but astronomically large. The next crucial question of the computational theory of error-driven learning thus concerns *efficiency*: is the number of errors not only finite but also *small* enough that the final grammar is efficiently converged on? Of course, the number of errors made by the learner should be allowed to grow with the complexity of the learning task. In constraint-based phonology, the simplest measure of complexity is the number of phonological constraints. Thus, the number of errors is "small" (and the learner efficient) provided it grows slowly (*polynomially*) with the number of constraints.

2.2. **Stochastic tolerance**

The implementation of error-driven learning schematized in (1) is called *deterministic*, to distinguish it from the *stochastic* implementation (Boersma 1997, 1998; Boersma and Hayes 2001; Coetzee and Pater 2008, 2011; Coetzee and Kawahara 2013; Jarosz 2013; Boersma and Pater to appear) schematized in (2). The latter differs because the current piece of data (say, datum 1) is compared not with the current grammar (say, G_1) but with a variant thereof (say, G'_1) sampled from a neighborhood of the current grammar (depicted as a dashed circle centered at G_1).

(2)



Assume that the stochastic error-driven learner is trained on a sequence of data generated by some (*deterministic*[4]) target grammar. In the worst-case scenario, the stochastic learner can make more errors than the deterministic learner, because the stochastic component can derail the leaner away from the most straightforward learning path (the simulations reported in section 5 will show that to indeed be the case). The worst-case number of errors made by a stochastic learner can thus be expressed as the sum of two terms. The first term (3a) is the worst-case number of errors made by the deterministic learner, which measures the difficulty of error-driven learning. The second term (3b) is the worst-case number of *additional* errors made by the stochastic learner, which measures the slow down due to the stochastic implementation.

(3)

$$\begin{array}{c}\text{worst-case number}\\ \text{of errors made by}\\ \text{the } \textit{stochastic} \text{ learner}\end{array} = \underbrace{\begin{array}{c}\text{worst-case number}\\ \text{of errors made by}\\ \text{the } \textit{deterministic} \text{ learner}\end{array}}_{\text{(a)}} + \underbrace{\begin{array}{c}\text{worst-case number}\\ \text{of additional errors due}\\ \text{to the stochastic implementation}\end{array}}_{\text{(b)}}$$

The next question of the theory of error-driven learning concerns its *tolerance* of the stochastic component: is the number (3b) of additional errors due to the stochastic implementation small enough not to disrupt efficiency? Again, this number (3b) of additional errors should be allowed to grow with the complexity of the learning task, measured as the number of constraints. Thus, the number (3b) of additional errors is "small' (and the learner *tolerant* of the stochastic component) provided it grows slowly (polynomially) with the number of constraints.[5]

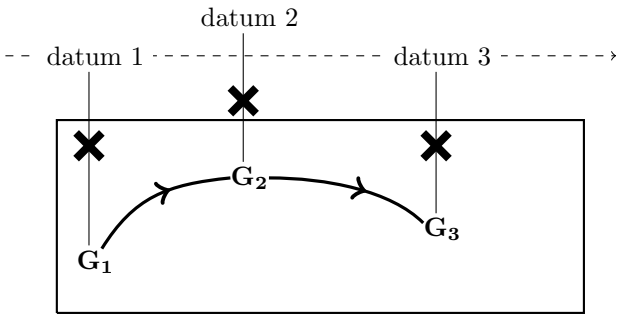
[4]General results on the ability of a stochastic error-driven learner to learn a *stochastic* target grammar are still unavailable (but see Boersma and Hayes 2001 for simulation results and Magri and Storme forth. for a detailed analysis of those simulation results).

[5] The number (3b) of additional errors is also expected to depend on the "size" of the stochastic component, namely the radius of the dashed circles in (2). A large radius means that the stochastic component can exert a large effect because the stochastic grammars $G'_1, G'_2, \ldots$ can be far apart from the deterministic grammars $G_1, G_2, \ldots$. Yet, the term (3b) will turn out to grow slowly with the size of the stochastic component in both the OT and HG implementations (see footnote 25). I thus only focus on its growth with the number of constraints, which instead will turn out to distinguish between the OT and HG learners.

2.3. **Noise robustness**

So far, I have assumed that the learner is trained on a stream of *pristine* data generated by some target grammar in the typology. A more realistic learning setting needs instead to allow for the possibility that a small portion of the training data could have been *corrupted* by transmission noise or speech errors (Gibson and Wexler 1994, p. 410; Frank and Kapur 1996, p. 625; Boersma and Hayes 2001, pp. 66-67; Bíró 2006). I schematize this more realistic noisy setting as in (4), where the data off the dotted line (such as datum 2) are meant to be corrupted by noise and thus inconsistent with the rest of the pristine data.

(4)



Suppose that the (possibly infinite) sequence of pristine training data is interspersed with a finite number of data corrupted by noise. No assumptions are made on the corrupted data apart from there being only a finite number of them.[6] The corrupted data make the learning problem harder (Magri 2013a, section 2) and might thus force the learner to make more errors. In other words, the number of errors made by the learner on a possibly corrupted training sequence can be expressed as the sum of two terms. The first term (5a) is the number of errors which would have been made on the sequence of pristine data with the corrupted data removed. The second term (5b) is the number of *additional* errors due to the corrupted data.

(5) worst-case number of errors made by the learner in the noisy setting = $\underbrace{\text{worst-case number of errors made in the noise-free case}}_{\text{(a)}}$ + $\underbrace{\text{worst-case number of additional errors due to the corrupted training data}}_{\text{(b)}}$

The final question of the theory of error-driven learning concerns its *robustness* to noise: is the number of additional errors (5b) due to the corrupted data small enough not to disrupt efficiency? Again, this number (5b) of additional errors should be allowed to grow with the complexity of the learning task, measured as the number of constraints (as well as, of course, the number of corrupted data and the size of the faulty updates they cause). Thus, the number (5b) of additional errors is "small' (and the learner *robust* to noise) provided it grows slowly (polynomially) with the number of constraints.

2.4. **Constraint-dependent and -independent perspective**

Tesar (2013, p. 18) summarizes the core tenet of generative linguistics as follows: "for language learning to be possible, the linguistic theory must have some kind of non-trivial structure [...] that can be exploited by a learner." What type of non-trivial structure yields guarantees for convergence, efficiency, stochastic tolerance, and noise robustness of error-driven learning? Within constraint-based phonology, "phonological theory contains two parts: a theory of substantive universals of phonological well-formedness, and a theory of formal universals of constraint interaction" (Prince and Smolensky 2004, p. 67). Correspondingly, computational guarantees for error-driven learning can be developed from either of two perspectives (Tesar and Smolensky 2000, section 1.2). The *constraint-independent* perspective derives computational guarantees from the formal properties of the mode of constraint interaction alone (say, the logic of transitive orders in the case of OT or the linear algebra underlying HG). It does not make any substantive assumptions on the phonological content of the constraints. The *constraint-dependent* perspective instead derives computational guarantees from the additional structure brought about through substantive restrictions on the

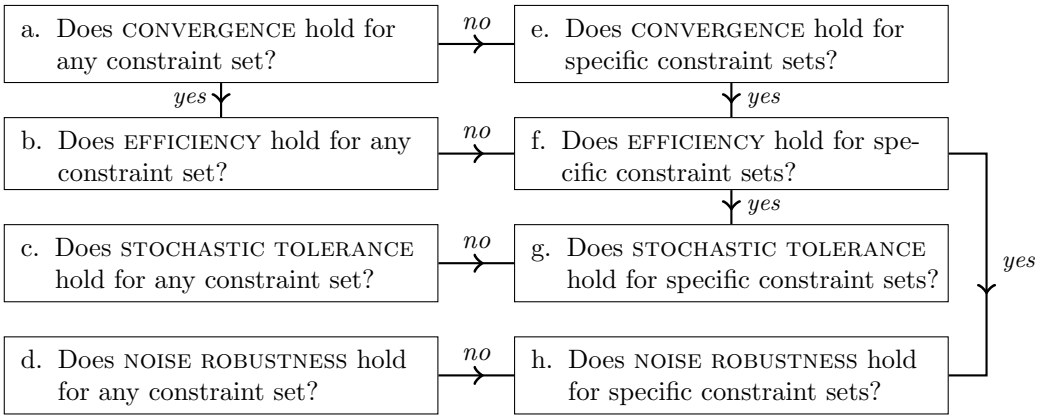
[6] Indeed, if an infinite number of corrupted data were allowed, the worst case number of errors would be infinite: whenever the learner rests on a hypothesis, it can be prompted to perform yet another update through a maliciously crafted piece of corrupted data.

phonological content of the constraints.[7] It is standard practice in the literature to adopt the constraint-independent perspective as the starting point, and to resort to the alternative constraint-dependent perspective only when the former fails. For instance, Eisner (2000, p. 23) writes: "The algorithms for learning [...] are designed to be general for any [constraint set] [...]. That is, these methods are not tailored (as others might be) to exploit the structure of some specific, putatively universal [constraint set]." The main reason for this privileged status accorded to the constraint-independent perspective is that, when it succeeds, it yields stronger results, namely computational guarantees that hold for a completely arbitrary constraint set. It thus circumvents the contentious issue of characterizing phonologically plausible constraint sets. And it ensures that the computational theory won't have to be revised whenever new phonological data are uncovered or better analyses developed.

2.5. **Summary**

The computational theory of error-driven learning in constraint-based phonology outlined in this section is synopsized in (6). The theory aims at computational guarantees for convergence, efficiency, stochastic tolerance, and noise robustness, which correspond to the four rows of the table. And it adopts a constraint-independent or -dependent perspective, which correspond to the two columns.

(6)



The horizontal arrows encode the privileged status accorded to the constraint-independent perspective: they say that a constraint-dependent question (listed in the right column) is tackled only when the corresponding constraint-independent question (listed in the left column) has been answered negatively. The vertical arrows encode additional dependencies among the eight questions. For instance, it only makes sense to investigate efficiency (i.e., whether the number of errors is small) provided convergence has been established (i.e., the number of errors is finite), as encoded by the two arrows between questions (6a) and (6b) and between (6e) and (6f). Furthermore, it only makes sense to investigate stochastic tolerance and noise robustness for a certain constraint set after efficiency has been established for that constraint set, as encoded with the two arrows between question (6f) and questions (6g) and (6h). In fact, a learner which is inefficient in its deterministic implementation and in a noise-free setting is useless to start with. On the other hand, even if efficiency fails from the ambitious constraint-independent perspective, it could still be the case that the number of additional errors (3b) and (5b) due to the stochastic implementation or the noisy data does remain small for any constraint set. There is therefore no arrow relating question (6b) with questions (6c) and (6d).

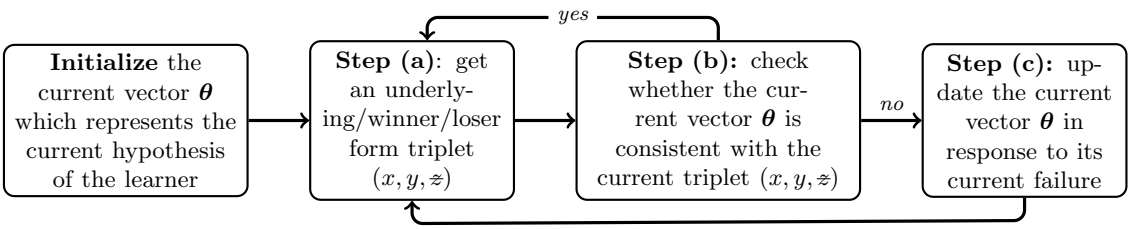
3. Error-driven learning in OT and HG

3.1. **Algorithmic core**

The HG typology of grammars is parameterized through a *weight* vector $\boldsymbol{\theta} = (\theta_1, \theta_2, \dots)$ which assigns a weight θ_k to each constraint C_k. The OT typology is instead parameterized through a constraint ranking. A ranking can be represented numerically through a *ranking* vector $\boldsymbol{\theta} = (\theta_1, \theta_2, \dots)$ which assigns a ranking value θ_k to each constraint C_k: a constraint is ranked above another constraint whenever the ranking value of the former is larger than the ranking value of the latter (Boersma 1997, 1998, 2009). The error-driven learning scheme sketched in (1) can then be formalized as in (7) within either HG and OT.

[7] The distinction between constraint-independent and -dependent perspective corresponds to the classical distinction between *formal* and *substantive* universals of linguistic theory (Chomsky 1965, §1.5).

(7)



The learner maintains a current vector $\boldsymbol{\theta}$ of weights or ranking values. For concreteness, I assume that its components are initially set all equal to zero. The current vector $\boldsymbol{\theta}$ is then updated by looping through steps (7a)-(7c). At step (7a), the learner receives the current piece of data. At step (7b), the learner checks whether the OT or HG grammar corresponding to the current vector $\boldsymbol{\theta}$ is consistent with that piece of data. If that is the case, the learner has nothing to learn from the current piece of data, loops back to step (7a) and waits for more data. Otherwise, the learner updates the current vector $\boldsymbol{\theta}$ at step (7c) and then loops back to step (7a) and starts all over again. The rest of this section describes steps (7a)-(7c).

3.2. **Data**

At step (7a), the learner receives a piece of data. At a minimum, this piece of data consists of a surface form y, say some form which is licit according to the phonotactics corresponding to the target grammar the learner is being trained on. In some applications, the corresponding underlying form x can be assumed to be provided as well. In other applications instead, the learner needs to be endowed with an additional subroutine to reconstruct the underlying form (e.g., set the current underlying form x identical to the current surface form y; Prince and Tesar 2004; Hayes 2004; Magri 2015b). Since this paper abstracts away from the proper definition of this subroutine, the underlying form is assumed to be provided along with the surface form at step (7a). The mapping (x, y) of the underlying form x to the surface form y must beat the mapping $(x, \not{z})$ of x to any other loser candidate $\not{z}$ (loser candidates are stricken out as a mnemonic). The learner thus needs to focus on one such loser candidate $\not{z}$. Usually, this loser candidate is chosen through a proper subroutine. This paper abstracts away from the proper definition of this subroutine as well. The loser form is thus assumed to be provided along with the underlying and winner forms at step (7a). In the end, the piece of data fed to the learner at step (7a) consists of a whole underlying/winner/loser form triplet $(x, y, \not{z})$.

3.3. **Update conditions**

At step (7b), the error-driven learner checks whether the HG or OT grammar corresponding to the current weight or ranking vector $\boldsymbol{\theta} = (\theta_1, \theta_2, \dots)$ is consistent with the current underlying/winner/loser form triplet $(x, y, \not{z})$, namely manages to make the current winner y beat the current loser $\not{z}$ for the underlying form x. In the case of HG, this consistency condition is (8). The *violation difference* corresponding to constraint C_k is the difference between the number of violations $C_k(x, \not{z})$ it assigns to the loser $\not{z}$ minus the number of violations $C_k(x, y)$ it assigns to the winner y. Condition (8) thus requires the average of the constraint violation differences weighted by the weights θ_k to be strictly positive.

$$(8)\quad \Big(\underbrace{C_1(x,\not{z}) - C_1(x,y)}_{\substack{\text{violation difference}\\ \text{of constraint } C_1}}\Big)\theta_1 + \Big(\underbrace{C_2(x,\not{z}) - C_2(x,y)}_{\substack{\text{violation difference}\\ \text{of constraint } C_2}}\Big)\theta_2 + \ldots > 0$$

In the case of OT, the consistency condition checked at step (7b) is (9). This condition requires the largest ranking value among the *winner-preferring* constraints in W (namely those constraints which have a positive violation difference, namely assign *less* violations to the intended winner y than to the intended loser $\not{z}$) to be larger than the largest ranking value among the *loser-preferring* constraints in L (namely those constraints which have a negative violation difference, namely assign *more* violations to the intended winner y than to the intended loser $\not{z}$).[8]

$$(9)\quad \max_{h\in W} \theta_h > \max_{k\in L} \theta_k$$

Conditions (8)/(9) will be referred to as the HG and OT *update conditions*, as the learner performs an update only in case these conditions fail. Note that both update conditions do not depend on the actual number of constraint violations but only on the constraint violation differences.

[8]If two or more ranking values tie, the ranking vector corresponds to multiple rankings, which resolve the ties in all possible ways while being consistent with the relative sizes of the ranking values. Condition (9) ensures that the OT grammar corresponding to each such ranking manages to make the winner y beat the loser $\not{z}$ (Boersma 2009).

3.4. **Update rules**

At step (7c), the error-driven learner updates the current weight or ranking vector $\boldsymbol{\theta} = (\theta_1, \theta_2, \dots)$ in case the corresponding grammar has failed on the current piece of data. In the case of HG, the literature has adopted the *Perceptron* (or *delta*) update rule in (10) (Jesney and Tessier 2011; Coetzee and Pater 2008, 2011; Coetzee and Kawahara 2013; Boersma and Pater to appear). If a constraint C_k is winner-preferring (loser-preferring), its violation difference $C_k(x, \text{ż}) - C_k(x, y)$ is positive (negative), so that its weight is increased (decreased) by (10). The algorithm (7) with the HG update condition (8) at step (7b) and the reweighing rule (10) at step (7c) is called the (*Perceptron*) HG *error-driven learning algorithm*[9]

(10) Update each current weight θ_k by adding the corresponding violation difference $C_k(x, \text{ż}) - C_k(x, y)$.

Despite its wide use in the current HG literature, this Perceptron update rule runs into the following problem. Constraints express penalties in constraint-based phonology, never rewards (Legendre et al. 2006; Keller 2000). Constraint weights must therefore be required to be nonnegative in order for HG to avoid undesired typological predictions. Yet, the Perceptron reweighing rule (10) does not in any way guarantee non negativity of the current and final weights entertained by the learner. A natural strategy to overcome this problem is to resort to the variant (11), which *truncates* the updates at zero (Boersma and Pater to appear, p. 19; Boersma and van Leussen 2014, section 5; Magri 2015a). The algorithm (7) with the HG update condition (8) at step (7b) and this new reweighing rule (11) at step (7c) is called the *truncated* Perceptron HG learner.

(11) Update each weight by adding the violation difference of the corresponding constraint as in (10), *unless that update would make that current weight negative, in which case do not modify that weight.*

Turning to OT, the generic re-ranking used by the error-driven learner at step (7c) has the shape in (12). The promotion component (12a) of the re-ranking rule promotes the winner-preferring constraints by a *promotion amount* which is equal to the inverse of the number w of winner-preferring constraints scaled by a constant c (Magri 2012b). This constant c used to *calibrate* the promotion amount on the number of winner-preferring constraints can be zero (in which case the learner performs no constraint promotion) or positive. In the rest of the paper, I will assume that c is larger than zero, so that the re-ranking rule performs both constraint demotion and promotion and thus comes closer to the reweighing rule (10)/(11) used by the HG learner. To illustrate, if $c = 1/2$ (this is the choice used in the simulations reported in the rest of the paper; see appendix E) and there are $w = 3$ winner-preferring constraints, each will be promoted by $c/w = 1/6$.

(12) a. Increase the ranking value of each of the w winner-preferring constraints by c/w;
b. decrease the ranking value of each *undominated* loser-preferring constraint by 1.

The demotion component (12b) of the re-ranking rule demotes by a fixed amount (say, 1) those loser-preferring constraints that really need to be demoted, namely those that are not currently ranked underneath a winner-preferring constraint and are therefore called *undominated* (Tesar and Smolensky 1998). The algorithm (7) with the update condition (9) at step (7b) and the re-ranking rule (12) at step (7c) is called a (*calibrated* and *gradual*) OT *error-driven ranking algorithm*. Note that both the HG reweighing rules (10)/(11) and the OT re-ranking rule (12) do not depend on the actual number of constraint violations but only on the constraint violation differences.

3.5. **Stochastic implementation**

At step (7b), the deterministic error-driven learner checks the update condition (8) or (9) for HG and OT, respectively. The only innovation of the stochastic implementation sketched in (2) is that the update condition is checked not for the current weights or ranking values $\theta_1, \theta_2, \dots$ but for their stochastic counterpart $\theta_1 + \epsilon_1, \theta_2 + \epsilon_2, \dots$, where $\epsilon_1, \epsilon_2, \dots$ are sampled independently from each other. The algorithm (7) with this stochastic update condition at step (7b) is called the OT/HG *stochastic* error-driven learner[10] (Boersma 1997, 1998; Boersma and Hayes 2001; Coetzee and Pater 2008, 2011; Coetzee and Kawahara 2013; Jarosz 2013; Boersma and Pater to appear). The stochastic values ϵ_k are usually sampled according to a gaussian distribution with zero mean and small variance. Since the tails of the gaussian distribution decrease exponentially fast, these stochastic values are bounded *with high probability* between some

[9] Boersma and Pater (to appear) call it *HG-GLA* instead. I prefer to keep the acronym "GLA" for a specific implementation of OT error-driven learning.

[10] The HG stochastic learner is called *Noisy HG-GLA* in Boersma and Pater (to appear). As explained in footnote 9, I prefer to reserve the acronym "GLA" to OT. Furthermore, I prefer "stochastic" over "noisy", in order to avoid any confusion between the stochastic implementation considered here and the noisy learning setting considered in subsection 2.3.

thresholds $-\Delta$ and $+\Delta$. From an analytical perspective, it is nonetheless convenient to assume they are *deterministically* bounded, namely sampled according to a distribution concentrated between $-\Delta$ and $+\Delta$. The analyses carry over with high probability to the gaussian distribution (Magri and Storme forth.).[11]

4. Convergence and efficiency

This section discusses some counterexamples where the number of errors made by the HG learner grows fast (exponentially) with the number of constraints, showing that efficiency does not hold constraint-independently, contrary to the case of the OT learner.

4.1. **A counterexample for the truncated Perceptron**

Prince and Smolensky (2004, section 10.2.2) put forward a simple counterexample against HG error-driven learning.[12] The counterexample is described in (13a) for an arbitrary number n of constraints and illustrated in (13b) for the case with $n = 5$ constraints.

(13) a.

	C_1	C_2	...	C_k	C_{k+1}	...	C_{n-1}	C_n
$(x,y,\tilde{z}_1)$	1	−2						
$(x,y,\tilde{z}_2)$		1	−2					
⋮			⋱	⋱				
$(x,y,\tilde{z}_k)$				1	−2			
⋮					⋱	⋱		
$(x,y,\tilde{z}_{n-2})$						1	−2	
$(x,y,\tilde{z}_{n-1})$							1	−2

b.

	C_1	C_2	C_3	C_4	C_5
$(x,y,\tilde{z}_1)$	1	−2			
$(x,y,\tilde{z}_2)$		1	−2		
$(x,y,\tilde{z}_3)$			1	−2	
$(x,y,\tilde{z}_4)$				1	−2

Let me unpack the notation. The n constraints $C_1, C_2, \dots, C_n$ are listed at the top of the matrix. The learner is trained on $n-1$ underlying/winner/loser form triplets $(x, y, \tilde{z}_i)$ corresponding to a single underlying form x, the winner candidate y, and $n-1$ loser candidates $\tilde{z}_1, \tilde{z}_2, \dots, \tilde{z}_{n-1}$. Since the update conditions and update rules described in section 3 only depend on the constraint violation differences, the counterexample is described by providing these violation differences for each constraint and each triplet (null violation differences are represented with empty entries for readability). Thus, the diagonal of entries equal to 1 in (13) say that constraint C_k assigns one more violation to the loser $\tilde{z}_k$ than to the winner y. The diagonal of entries equal to -2 say that constraint C_{k+1} assigns two more violations to the winner y than to the loser $\tilde{z}_k$. All other violation differences are null.

For concreteness, focus on the case with $n = 5$ constraints in (13b). The bottom row of the matrix requires the weight of constraint C_4 to be more than twice larger than the weight of constraint C_5. The penultimate row requires the weight of constraint C_3 to be more than twice larger than the weight of constraint C_4 and thus more than $2^2 = 4$ times larger than the weight of constraint C_5. And so on until the top row, which requires the weight of constraint C_1 to be more than $2^4 = 16$ times larger than the weight of constraint C_5. In other words, the weight of constraint C_1 starts null and has to climb to a final value 2^{n-1} which grows exponentially in the number n of constraints. Yet, the weights are incremented only by 1 with each update, since the positive violation differences in (13) are all equal to 1. We thus expect that a number of updates exponential in n is needed in order to converge to a final weight vector which assigns a sufficiently large weight to constraint C_1.

This prediction is borne out in the case of the HG error-driven learner with the truncated Perceptron reweighing rule (11). Indeed, I have run this learner for ten times on the counterexamples (13) corresponding to a number n of constraints between $n = 5$ and $n = 10$; see table 1 in appendix E for simulation results. The solid line in (14a) plots the largest number of errors (vertical axis) made by the HG learner over the ten runs as a function of the number n of constraints (horizontal axis).[13] Comparison with the function $162 \times 4^{n-5}$ plotted by the dashed line shows that the number of errors grows fast (exponentially) with the number n of constraints.[14] Indeed, the number of errors grows so fast that the learner makes more than 180,000 errors in the case with just ten constraints. The solid line in (14b) plots the final weight

[11] This constant Δ corresponds to the radius of the dashed circles in (2) and will therefore be referred to as the *size* of the stochastic component. The larger Δ, the larger the size of $\epsilon_1, \epsilon_2, \dots$, the larger the difference between the actual values $\theta_1, \theta_2, \dots$ and their stochastic counterparts $\theta_1 + \epsilon_1, \theta_2 + \epsilon_2, \dots$.

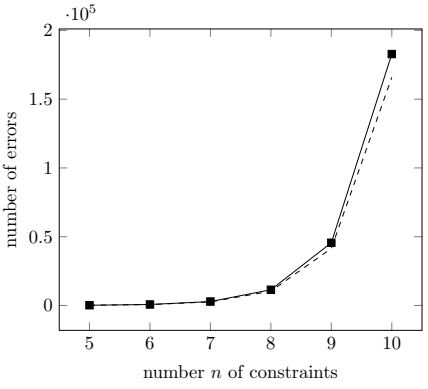
[12] Thanks to Paul Boersma (p.c.) for pointing this out to me.

[13] Here and throughout, I plot the *largest* number of errors over ten runs (rather than, say, the mean or the smallest number of errors) because the perspective adopted in this paper is that of bounds on the *worst-case* number of errors. In any case, the plot of the *smallest* number of errors would be almost indistinguishable; see subsection 4.8 for discussion.

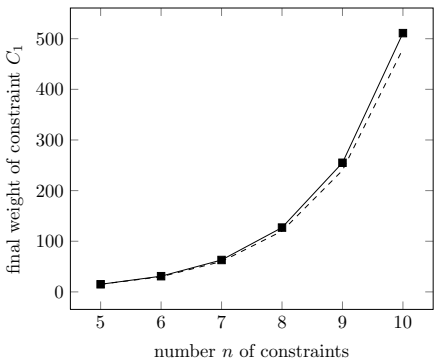
[14] The number of errors is even larger when the negative entries are smaller than -2; see table 2 in appendix E for simulation results when they are all equal to -3. Prince and Smolensky actually seem to assume that the size of the negative

of constraint C_1 (which is constant across the ten runs) as a function of the number n of constraints. Comparison with the function $15 \times 2^{n-5}$ plotted by the dashed line shows that this final weight grows fast (exponentially) with the number n of constraints, as predicted by Prince and Smolensky's reasoning.

(14) a.



b.



The counterexample (13) thus shows that no constraint-independent guarantees are possible for the efficiency of the HG error-driven learner with the *truncated* Perceptron reweighing rule. Any such guarantees will have to be restricted through specific assumptions on the constraint set.

Yet, this counterexample does not defeat the HG learner with the *original* Perceptron reweighing rule (10). Indeed, that learner converges after only $n-1$ updates (thus, nine updates suffice in the case with $n = 10$ constraints); see table 3 in appendix E for simulation results. The final weights do not depend on the number n of constraints and are small: the final weight of constraint C_1 is always equal to 1, the final weight of constraint C_n is always equal to -2, and the other final weights are all equal to -1. By resorting to negative weights, the original Perceptron thus defies Prince and Smolensky's challenge. How should we interpret this drastically different behavior of the original and the truncated Perceptron on the counterexample (13)? Does it show that the two learners have substantially different properties and that HG's assumption of nonnegative weights is particularly vexing from a learnability perspective? Or is it possible to twist the counterexample to extend it to the original Perceptron reweighing rule, thus showing that the two implementations are on a par? The latter turns out to be the case.

4.2. **A counterexample for both the original and the truncated Perceptron**

Consider the counterexample described in (15a) for an arbitrary number n of constraints and illustrated in (15b) for the case with $n = 5$ constraints. There are two differences with the counterexample considered in the preceding subsection (see subsections 4.6-4.7 for discussion). The first difference is that each negative entry is followed by two positive entries both equal to 1, but for the penultimate row (where we can fit only one positive entry after the negative entry) and the last row (where we can fit none). Thus, all rows have three positive entires, apart from the penultimate row (which has only two positive entries) and the last row (which has only one). The second difference is that the size of the negative entries is readjusted, so that it is equal to the number of positive entries per row. Thus, the negative entry of each row is equal to -3 apart from the penultimate row (whose negative entry is equal to -2) and the last row (whose negative entry is equal to -1).

(15) a.

$$\begin{array}{c} \begin{array}{cccccccccccccccc} C_1 & C_2 & C_3 & C_4 & C_5 & \dots & C_k & C_{k+1} & C_{k+2} & C_{k+3} & \dots & C_{n-3} & C_{n-2} & C_{n-1} & C_n \end{array} \\ \left[\begin{array}{cccccccccccccccc} 1 & -3 & 1 & 1 & & & & & & & & & & & \\ & 1 & -3 & 1 & 1 & & & & & & & & & & \\ & & & & & \ddots & & & & & & & & & \\ & & & & & & 1 & -3 & 1 & 1 & & & & & \\ & & & & & & & & & & \ddots & & & & \\ & & & & & & & & & & & 1 & -3 & 1 & 1 \\ & & & & & & & & & & & & 1 & -2 & 1 \\ & & & & & & & & & & & & & 1 & -1 \end{array}\right] \end{array}$$

b.

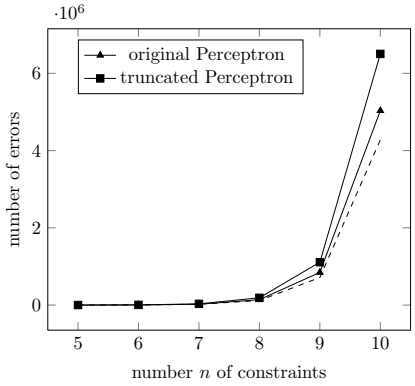
$$\begin{array}{c} \begin{array}{ccccc} C_1 & C_2 & C_3 & C_4 & C_5 \end{array} \\ \left[\begin{array}{ccccc} 1 & -3 & 1 & 1 & \\ & 1 & -3 & 1 & 1 \\ & & 1 & -2 & 1 \\ & & & 1 & -1 \end{array}\right] \end{array}$$

The solid lines in (16a) plot the largest number of errors made by the HG error-driven learner with the two reweighing rules (10) and (11) over ten runs on this counterexample (15) corresponding to a number

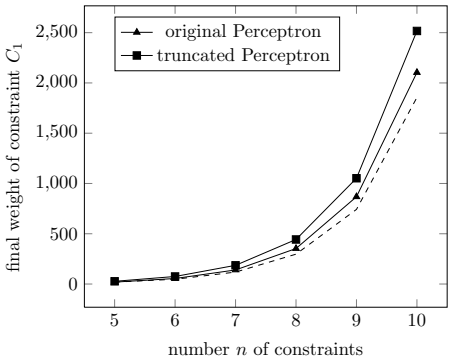
entries grows with the number n of constraints (they consider the case with $n = 10$ constraints and assume negative entries equal to -10). But unbounded constraint differences are not necessary, as shown by the simulation results reported here.

n of constraints between $n = 5$ and $n = 10$; see tables 4 and 5 in appendix E for simulation results. Comparison with the function $550 \times 6^{n-5}$ plotted by the dashed line shows that the number of errors grows fast (exponentially) with the number n of constraints for both reweighing rules. Indeed, the HG learner with the original (truncated) reweighing rule makes over five million (six million) errors in the case with just ten constraints! The solid line in (16b) plots the final weight of constraint C_1 in the longest of the ten runs as a function of the number n of constraints. Comparison with the function $19 \times 2.5^{n-5}$ plotted by the dashed line shows that this final weight again grows fast (exponentially) with the number n of constraints.

(16) a.



b.



I conclude that the counterexample (15) shows that no constraint-independent efficiency guarantees are possible in the case of the HG error-driven learner, independently of the issue of non-negative weights and thus of the choice between the original and the truncated Perceptron reweighing rule. Any efficiency guarantees will have to be restricted through specific assumptions on the constraint set. The rest of this section discusses the counterexample (15).

4.3. **Discussion/I: the margin**

The HG error-driven learner with the Perceptron reweighing rule (10) converges (Boersma and Pater to appear, building on guarantees for the Perceptron algorithm: Block 1962; Novikoff 1962; Rosenblatt 1958, 1962; Minsky and Papert 1969; Cristianini and Shawe-Taylor 2000, Theorem 2.3; Cesa-Bianchi and Lugosi 2006, ch. 12; Mohri et al. 2012, ch. 7). Whenever trained on data consistent with some HG grammar and with bounded violation differences, this HG learner can only make a finite number of errors, which can be bounded as in (17), in terms of the number of constraints and a quantity that captures a property of the training data, namely their *margin (of separability)*.[15]

$$\text{(17)}\quad \text{number of errors} \lessapprox \frac{\text{number of constraints}}{\big(\text{margin of the training data}\big)^2}$$

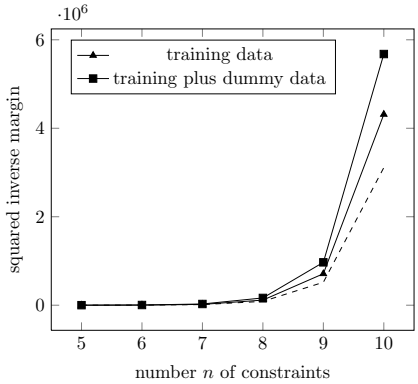
The precise definition of the *margin (of separability)* which appears in the denominator of the error-bound is somewhat involved and therefore relegated to appendix A. The following intuitive illustration suffices for the rest of the paper. The error-bound holds under the assumption that the training data are consistent with some HG grammar. Yet, consistent training data can differ in their *degree of consistency*. The data have a *high* degree of consistency if they are consistent with a certain HG grammar and remain consistent when the corresponding weight vector is tampered with. The data have instead a *small* degree of consistency if even a slight modification of any consistent weight vector affects consistency. The margin which appears in the denominator of (17) can be interpreted as the degree of consistency of the training data. Intuitively, training data with a large degree of consistency should require less updates: it should be easier to shoot at a consistent weight vector. While training data with a small degree of consistency should require more updates: a careful aim is required to shoot precisely at a consistent weight vector. The error-bound (17) formalizes this intuition: training data with a high (low) degree of consistency have a large (small) margin, yielding a small (large) error-bound (17) which provides guarantees for a better (worse) performance of the HG learner.

As the number of constraints grows, the constraint interactions can become more subtle and delicate, the slackness in the choice of the weights thus decreases, shrinking the degree of consistency of the data,

[15] To keep the presentation lean, I only provide approximate expressions of the error-bounds, ignoring most of the numerical constants. For this reason, I use "$\lessapprox$" rather than "$<$". For the exact expression of the OT error-bounds recalled in this paper, see Magri (2015c); for an accessible presentation of the exact expression of the HG error-bounds, see Magri (2015a).

namely their margin. If the margin in the denominator of (17) shrinks fast (exponentially) with the number of constraints, it trumps the slow (linear) growth of the numerator. The overall error-bound thus grows fast (exponentially) and provides no guarantees on the efficiency of the HG learner. This is precisely what happens in the case of the counterexample (15) — analogous considerations hold for the original counterexample (13). The inverse of the squared margin is plotted by the triangles in (18), as computed in appendix B. Comparison with the function $400 \times 6^{n-5}$ plotted by the dashed line shows that it grows fast (exponentially) with the number n of constraints.

(18)



In conclusion, the error-bound (17) settles question (6a) concerning constraint-independent convergence of the HG Perceptron learner. But because of its dependence on the margin, that bound does *not* settle question (6b) concerning constraint-independent efficiency. That question is answered negatively by counterexamples such as the one in subsection 4.2, which shows that constraint-independent efficiency guarantees are impossible.

These considerations extend from the original to the truncated Perceptron reweighing rule (11). To illustrate the intuition, suppose that at a certain iteration the original Perceptron demotes the weight of some constraint C_k from 0 to the forbidden negative value -1. The truncated Perceptron is unable to mimic that demotion and will thus leave the weight of C_k at 0. Yet, the learning dynamics of the two algorithms can be realigned by assuming that, immediately after that offending demotion, the original Perceptron is fed with a piece of *dummy data* whose constraint violation differences are all null apart from the one corresponding to C_k which is instead equal to $+1$. Since the current weight of C_k is negative, the original Perceptron will perform an update in response to this piece of dummy data. This update promotes back the weight of C_k to the value 0 where the truncated Perceptron had left it. A formalization of this reasoning (Magri 2015a) shows that the error-bound (17) for the original Perceptron extends to the truncated variant, only with the margin of the training data replaced by the margin of the training plus dummy data, as in (19).

$$\text{(19)}\quad \text{number of errors} \lessapprox \frac{\text{number of constraints}}{\big(\text{margin of the training plus dummy data}\big)^2}$$

The inverse of the squared margin of the training plus dummy data in the case of the counterexample (15) is plotted by the squares in (18). Again, it grows fast (exponentially) in the number of constraints, so that the error-bound (19) for the truncated Perceptron HG learner only guarantees constraint-independent convergence, not efficiency.

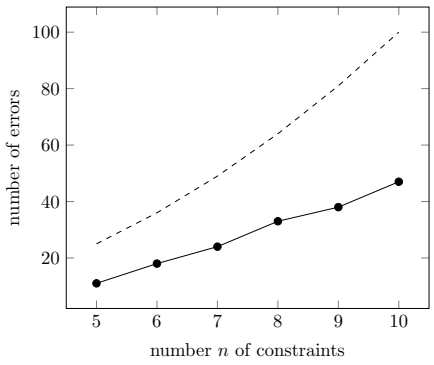
Extending the training data with additional data can only decrease the degree of consistency, namely the margin. Thus, the margin of the training data extended with the dummy data which appears in the error-bound (19) for the truncated Perceptron should be equal to or smaller than the margin of just the original data which appears in the error-bound (17) for the original Perceptron. The gap between the two solid lines in (18) quantifies the difference between the two margins in this case. Thus, the error-bound (19) for the truncated Perceptron is worse than (namely, at least as large as) the error-bound (17) for the original Perceptron. Indeed, the truncated Perceptron performs more errors in the simulations plotted in (16a). The difference between the two margins quantifies the price that needs to be paid for HG's assumption of non-negative weights.

4.4. **Discussion/II: comparison with OT**

The solid line in (20) plots the largest number of errors made by the OT learner over ten runs on the counterexample (15) corresponding to a number n of constraints between $n = 5$ and $n = 10$; see appendix E for simulation details and in particular table 6 for simulation results. Comparison with the function n^2

plotted by the dashed line[16] shows that the number of errors grows slowly (sub-quadratically) with the number n of constraints, so that the learner makes less than fifty errors in the case with ten constraints.[17]

(20)



These simulation results are not surprising. In fact, suppose that the OT learner does not promote "too much", namely it adopts a promotion component (12a) corresponding to a calibration constant c which is not "too large", namely strictly smaller than 1. Whenever trained on data consistent with some OT grammars, this OT learner converges and the number of errors is bounded by (21), in terms of the number of constraints and the calibration constant c used in the definition of the promotion amount (Tesar and Smolensky 1998; Boersma 1997, pp. 323–327; Boersma 2009; Magri 2012b,a).

$$\text{(21)}\quad \text{number of errors} \lessapprox \frac{(\text{number of constraints})^2}{1-c}$$

This error-bound (21) depends on the number of constraints only through its numerator. Thus, it grows slowly (quadratically) in the number of constraints. Furthermore, this error-bound requires no assumptions on the constraints. In conclusion, this error bound provides a positive answer to questions (6a) and (6b) concerning constraint-independent efficient convergence of the deterministic OT error-driven learner.

4.5. **Discussion/III: the OT and HG typologies coincide**

The typology of grammars[18] predicted by HG can be larger than the one predicted by OT, because only HG allows *gang-up* effects. Could the slowness of the HG learner on the counterexample (15) relative to the OT learner be due to the fact that the former is exploring a larger typology of grammars? The answer is negative: the counterexample has been constructed in such a way that the HG and OT typologies exactly coincide. To illustrate, consider the case with $n = 5$ constraints in (15b). This matrix of constraint violation differences corresponds, for instance, to the constraint violation profiles in (22). To illustrate, the topmost and leftmost entry in the matrix (15b) is 1 because tableau (22) says that constraint C_1 assigns one violation to the loser $\not{z}_1$ and none to the winner y

(22)

Input: x		C_1	C_2	C_3	C_4	C_5
a.	y		***	***	**	*
b.	z_1	*		****	***	*
c.	z_2		****		***	**
d.	z_3		***	****		**
e.	z_4		***	***	***	

The OT and HG typologies of grammars (computed with `OT-Help`; Staubs et al. 2010) corresponding to the constraint set described in (22) coincide: they both consist of five grammars, one for each mapping of the

[16]The function n^2 has been chosen here because the exact version of the OT error bound (21) is $\frac{1}{2(1-c)}n(n-1)$ (Magri 2012a), which boils down to $n(n-1) \sim n^2$ when $c = 1/2$, as in the simulations reported here.

[17]In the case of the HG learner (with either reweighing rule), the largest final weight (namely the weight of constraint C_1) grows fast (exponentially) with the number of constraints, as plotted in (16b). In the case of the OT learner instead, the largest final ranking value (again, the one of constraint C_1) remains constant (always equal to 0.17), as shown by table 6 in appendix E.

[18]As it has become standard practice in computational phonology (Heinz 2011), I am using the term *grammar* to denote a mapping from underlying to surface representations. A grammar is thus independent of the choice of the grammatical framework (e.g., the choice between OT and HG). In this sense, OT rankings and HG weights are not grammars, but framework-specific *parameterizations* of the typology of grammars.

underlying form to one of the five candidates. This conclusion extends from the case with $n = 5$ constraints to the case with an arbitrary number n of constraints: the constraint violation differences described in (15a) correspond to a set of n constraints whose corresponding OT and HG typologies coincide. The inefficiency of the HG learner on the counterexample thus cannot be due to the fact that it is exploring a larger typology. In other words, the counterexample shows that, even in those cases where the typologies predicted by OT and HG turn out to be identical (as recently argued to be often the case in Pater 2009), the OT parameterization of the typology outperforms the HG one from the perspective of error-driven learning.

4.6. **Discussion/IV: the two crucial properties of the counterexample**

The counterexample (15) has two properties which are both crucial in order to slow down the HG learner. The *first property* is that the size of each negative entry is equal to the number of positive entries in the corresponding row (for instance, the negative entry of the penultimate row is equal to -2 because that row has two positive entries). The *second property* is that each negative entry is followed by two positive entries equal to 1 (but for the last two rows, where there is not enough space to fit in these two extra positive entries). To see that the first property is indeed crucial, consider, say, the variant (23) where the negative entries are equal to -3 throughout, rather than being equal to -2 and -1 in the bottom two rows. There is no slow down in this case: with $n = 10$ constraints, the HG Perceptron learner makes only 206 errors, contrary to the five million errors made on the original counterexample (15); see table 7 in appendix E for simulation results.

(23) a.

C_1	C_2	C_3	C_4	C_5	...	C_k	C_{k+1}	C_{k+2}	C_{k+3}	...	C_{n-3}	C_{n-2}	C_{n-1}	C_n
1	−3	1	1											
	1	−3	1	1										
					⋱									
						1	−3	1	1					
										⋱				
											1	−3	1	1
												1	−**3**	1
													1	−**3**

b.

C_1	C_2	C_3	C_4	C_5
1	−3	1	1	
	1	−3	1	1
		1	−**3**	1
			1	−**3**

To see that the second property of the counterexample (15) is crucial as well, consider the variant (24) where each negative entry is followed by just *one* positive entry. Again, there is no slow down in this case: with $n = 10$ constraints, the Perceptron HG error-driven learner makes only 133 errors, again way less than the five million errors made on the original counterexample (15); see table 8 in appendix E for simulation results.

(24) a.

C_1	C_2	C_3	C_4	C_5	...	C_k	C_{k+1}	C_{k+2}	C_{k+3}	...	C_{n-3}	C_{n-2}	C_{n-1}	C_n
1	−3	1	**0**											
	1	−3	1	**0**										
					⋱									
						1	−3	−1	**0**					
										⋱				
											1	−3	1	**0**
												1	−2	1
													1	−1

b.

C_1	C_2	C_3	C_4	C_5	C_6
1	−3	1	**0**		
	1	−3	1	**0**	
		1	−3	1	**0**
			1	−2	1
				1	−1

Why are these two properties crucial in order to defeat the HG learner? The next subsection provides some intuition.

4.7. **Discussion/V: explaining the two crucial properties of the counterexample**

As just noted, one crucial property of the counterexample (15) is that the size of the negative entries is always equal to the number of positive entries in the corresponding row. This property ensures that gang-up effects are impossible in the case of the counterexample, as explained in appendix C (this is a special case of an argument from Magri 2013b, section 4). The OT and HG update conditions thus collapse: whenever the OT learner would update the current vector $\boldsymbol{\theta}$, the HG learner would update it as well. Furthermore, suppose that the OT learner uses a promotion amount of the form (12a) corresponding to the choice $c = 1$ of the calibration constant c. Because of this special choice $c = 1$ of the calibration

constant, it turns out that any update made by the OT learner coincides with the update made by the HG learner with the original Perceptron reweighing rule, as explained in appendix C.[19] We have thus obtained the following conclusion: since the size of the negative entries in the counterexample (15) is always equal to the number of positive entries, the HG learner with the original Perceptron reweighing rule mimics the OT learner with the calibration constant c set equal to $c = 1$. In the sense that, whenever the OT learner trained on the counterexample makes an update, the HG learner trained on the same sequence of data makes an update as well and furthermore the OT and HG updates exactly coincide. In order for this conclusion to hold, the size of the negative entries needs to be exactly equal to the number of positive entries. For instance, the HG learner does not mimic the OT learner on the variant in (23), where the size of the negative entries in the last two rows is different from the number of positive entries (see appendix C for a counterexample).

Since the HG learner trained on the counterexample (15) mimics the OT learner with the choice $c = 1$ for the calibration constant c, we are prompted to look into the behavior of this specific OT learner. Magri (2012b, section 7.5) reported that the number of errors made by this specific OT learner on the counterexample (15) grows fast (exponentially) in the number of constraints. This fact is not surprising. Since Tesar and Smolensky (1998, section 2.4), there has been a consensus in the OT learnability literature that constraint demotion works fine while constraint promotion is dangerous in OT. The promotion amount (or, equivalently, the calibration constant c used to define it) should thus be either null or at least small. It turns out that it is sufficient and necessary for the calibration constant c to be strictly smaller than 1 in order for efficient convergence to hold, as guaranteed by the error bound (21). Indeed, even a calibration constant c equal to $c = 1$ (the smallest forbidden value) might cause the OT learner to make too many errors, as shown by this counterexample (15).

The failure of the OT learner with the calibration constant c set equal to the forbidden value $c = 1$ on the counterexample (15) is due to the second property of the counterexample singled out in subsection 4.6, namely the fact that each negative entry is followed by two positive entries (but for the last two rows). In fact, the OT learner converges quickly on the case (24) with less additional positive entries, even with the calibration constant c set equal to the forbidden value $c = 1$ (see Magri 2013b, section 4.8, corollary 2). It is not hard to grasp the intuitive reason why the OT learner is defeated by the two additional positive entries at the right of each negative entry. In fact, each of these additional positive entries makes the corresponding constraint winner-preferring. Yet, these additional winner-preferring constraints contribute nothing to consistency: the counterexample (15) is only consistent with the ranking $C_1 \gg C_2 \gg \ldots \gg C_n$, just as the variant (24) with less winner-preferring constraints. These additional winner-preferring constraints are thus only a distraction for an OT learner which is too sensitive to them because it performs too much constraint promotion, namely adopts too large a calibration constant $c = 1$. The two layers of additional useless winner-preferring constraints of the counterexample (15) suffice to distract the OT learner enough to compromise its efficiency.

Let me take stock. The counterexample (15) has two crucial properties, concerning the size of the negative entries (which is equal to the number of positive entries) and the distribution of the positive entries (there are two after each negative entry). The former property of the counterexample ensures that the Perceptron HG learner mimics the OT learner with a calibration constant c which is too large because equal to the forbidden value $c = 1$. The latter property of the counterexample ensures enough useless winner-preferring constraints to defeat an OT learner which is too sensitive to them because it sets the calibration constant equal to the forbidden value $c = 1$ and thus promotes too much. In conclusion, the HG learner performs too many updates on the counterexample (15) because it performs an update whenever the OT learner performs an update; and the latter performs too many updates because it promotes too much. In other words, HG error-driven learning is as bad an idea from a constraint-independent perspective as OT error-driven learning with a large promotion component, in defiance of Tesar and Smolensky's warning.[20]

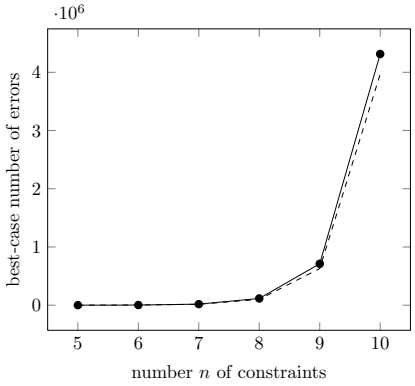
[19]This claim is actually only true for a slight variant of the Perceptron reweighing rule (10), whereby the constraint differences are rescaled by a (nonnegative) constant called a *step size* (or *plasticity*) before being used to additively update the current weights. The step sizes change nothing in the behavior and the analysis of the HG learner.

[20]An anonymous reviewer raises the following question: if the problem with the HG learner on the counterexample (15) is that it is equivalent to an OT learner which promotes too much, couldn't the problem with the HG learner be solved simply by promoting less, exactly as prescribed by the careful calibration (12a) of the promotion amount in the case of the OT learner? The problem with this proposal is that it implies abandoning the general convergence theory recalled in subsection 4.3. To the best of my knowledge, no such simple fix has been shown to be viable in the extensive literature on the Perceptron algorithm.

4.8. **Discussion/VI: the HG learner is slow even on the most favorable training sequence**

The solid lines in (16) plot the largest number of errors made by the HG learner over ten runs on the counterexample (15). These simulation results leave open the possibility that there exist some particularly favorable sequences of training data where the learner manages to converge after a smaller number of errors. Such a scenario would leave open the possibility of constraint-independent efficiency guarantees, through proper restrictive assumptions on the training sequences. Yet, this possibility can be ruled out as well. In fact, appendix D shows how to compute analytically a *lower bound* on the *best-case* number of errors made by the HG Perceptron learner:[21] it guarantees that the number of errors needed to reach convergence can never be smaller than a certain quantity, not even on the most favorable training sequence. This lower-bound on the best-case number of errors is plotted by the solid line in (25) as a function of the number n of constraints between $n = 5$ and $n = 10$.

(25)



Comparison with the function $400 \times 6.3^{n-5}$ plotted by the dashed line shows that also the *best-case* number of errors on the counterexample grows fast (exponentially) with the number n of constraints. Furthermore, comparison with the actual number of errors plotted in (16) shows that the very large number of actual errors in the simulations is very close to the best-case number of errors and thus cannot be due to particularly unfavorable learning sequences.

4.9. **Discussion/VII: the Perceptron in Machine Learning and Computational Linguistics**

How do the counterexamples presented in this section square with the good performance of the Perceptron reported in the machine learning and computational linguistics literature? For instance, to advocate the use of the Perceptron in HG, Boersma and Pater (to appear) mention its successful application in computational linguistics (Collins 2002). Furthermore, Mohri et al. (2012, p. 168) state that "the kernel Perceptron algorithm [...] is commonly used [...] in a variety of applications" in machine learning. Indeed, Pater (2009, p. 1021) advocates the HG implementation of constraint-based phonology in particular because it can rely on this computational linguistics and machine learning literature: "one broad argument for weighted constraints [...] is that weighted constraints are compatible with existing well-understood algorithms for learning variable outcomes and for learning gradually."

No contradiction subsists between the counterexamples presented in this section and the results reported in the machine learning and computational linguistics literature on the Perceptron. In fact, the latter literature has looked at a variant of the Perceptron called the *kernel (dual) Perceptron*. This is indeed the variant of the Perceptron mentioned in the preceding quote from Mohri et al. (2012) and tested in Collins (2002). This variant outperforms the (*primal*) Perceptron considered here at the expense of a much richer representation of the learning history. In fact, the primal Perceptron maintains an impoverished summary of its past learning history: it does not keep track of the previous updates nor of the training data that triggered those updates. The kernel dual Perceptron instead stores the (finite) set of training data and maintains a counter (called a *dual variable*) of the number of updates triggered by each piece of data up to the current iteration. Because of this richer representation of the learning history, the dual Perceptron does not fit into the error-driven learning scheme investigated in this paper (see footnote 2). The conclusions reached in the machine learning and computational linguistics literature concerning the kernel dual Perceptron are therefore irrelevant to the topic of this paper, namely the development of proper error-driven learners.

[21] The technique used in appendix D does not extend straightforwardly to the *truncated* variant.

4.10. **Summary**

This section has compared the OT and HG implementations of error-driven learning from the perspective of constraint-independent efficiency: how fast does the worst-case number of errors grow with the number of constraints? The error-bound for the OT implementation grows slowly with the number of constraints and thus provides constraint-independent efficiency guarantees. The error-bound for the HG implementation instead depends on the margin and can therefore grow fast with the number of constraints. It is indeed possible to construct cases where the HG and OT typologies coincide and yet the HG learner makes over five million errors with just ten constraints, which compares poorly with the less than fifty errors made by the OT learner. These simulation results show that no constraint-independent efficiency guarantees are possible for the HG learner.

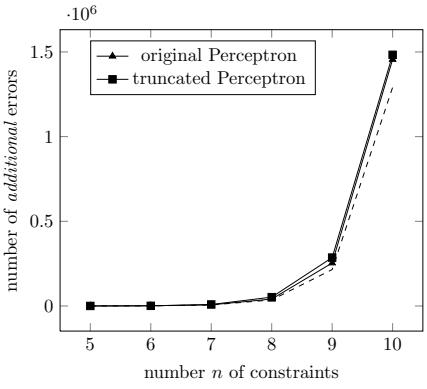
5. Stochastic tolerance

This section discusses some counterexamples where the number of additional errors made by the HG learner because of the stochastic component grows fast (exponentially) with the number of constraints, showing that the learner does not tolerate the stochastic implementation constraint-independently, contrary to the case of the OT learner.

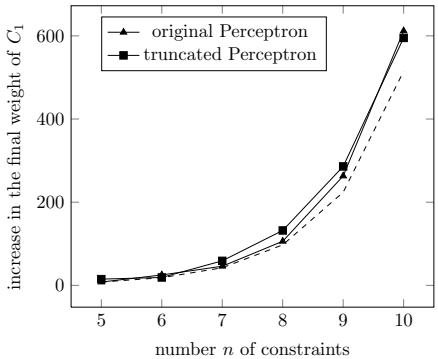
5.1. **Counterexample**

Consider again the counterexample (15) constructed in subsection 4.2. For each choice of the number n of constraints between $n = 5$ and $n = 10$, I have run the deterministic and stochastic HG learner ten times, with both the original and the truncated Perceptron reweighing rule; see appendix E for simulation details and in particular tables 4 and 5 for simulation results. The two solid lines in (26a) plot the difference between the largest number of errors made by the stochastic learner with the two reweighing rules over the ten runs minus the number of errors made by the deterministic learner on that same training sequence. In other words, they plot the additional number of errors due to the stochastic component. Comparison with the function $166 \times 6^{n-5}$ plotted by the dashed line shows that the number of additional errors grows fast (exponentially) with the number n of constraints. Indeed, in the case with just ten constraints, the stochastic learner performs almost one million and a half additional errors compared with the deterministic learner, which in turn was already performing over five million errors! For completeness, the solid lines in (26b) plot the difference between the final weight of constraint C_1 entertained by the stochastic learner with the two reweighing rules at the end of the longest run minus the final weight of C_1 entertained by the deterministic learner at the end of that same run. In other words, they plot the increase in the final weight of constraint C_1 due to the stochastic component. Comparison with the function $8 \times 2.3^{n-5}$ plotted by the dashed line shows that the increase grows fast (exponentially) with the number n of constraints.

(26) a.



b.



In conclusion, this counterexample shows that it is not possible to develop constraint-independent guarantees that the HG error-driven learner (with either reweighing rule) tolerates the stochastic implementation, namely that the number of additional errors due to the stochastic component remains small. Any such guarantees will crucially have to be restricted by specific assumptions on the constraint set.

5.2. **Discussion/I: the margin, again**

The HG error-driven learner with the stochastic update condition and the original Perceptron reweighing rule (10) converges (Boersma and Pater to appear). Whenever trained on data consistent with some (deterministic) HG grammar and with bounded constraint differences, it can only make a finite number of errors, which can be bounded as in (27), in terms of the number of constraints and the margin of the

training data.[22] The same error-bound holds for the truncated Perceptron reweighing rule (11), only with the margin of the training plus the dummy data (rather than the margin of only the training data) at the denominator (Magri 2015a).

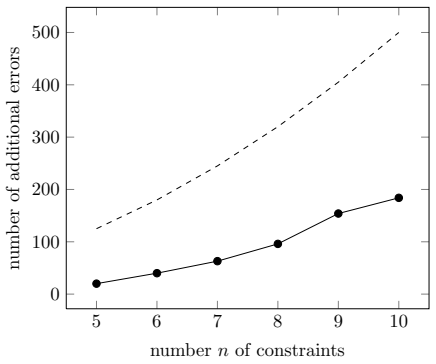
$$(27)\quad \text{number of errors} \lessapprox \underbrace{\frac{\text{number of constraints}}{\big(\text{margin of the training data}\big)^2}}_{\text{(a)}} + \underbrace{\frac{\Delta \times \text{number of constraints}}{\big(\text{margin of the training data}\big)^2}}_{\text{(b)}}$$

This error-bound (27) is the sum of two terms. The first term (27a) coincides with the error-bound (17) for the deterministic HG learner. The second term (27b) thus quantifies the additional number of errors due to the stochastic implementation. This latter term consists of the number of constraints (scaled by the size Δ of the stochastic component) divided by the squared margin. This term (27b) is large (namely, provides a weak bound) when the margin is small. This makes intuitive sense, given that the margin represents the degree of consistency of the data: a small margin means that there is little slackness in the choice of consistent weights so that the perturbation of the current weights due to the stochastic component can hold sway. The margin of the counterexample (15) decreases fast (exponentially) with the number of constraints, so that its squared inverse grows fast, as plotted in (18). The bound (27b) on the number of additional errors thus grows fast and does not settle question (6c) concerning constraint-independent stochastic tolerance. That question is answered negatively by the counterexample described in subsection 5.1, which shows that constraint-independent guarantees of stochastic tolerance are impossible in HG.

5.3. **Discussion/II: comparison with OT**

The solid line in (28) plots the difference between the largest number of errors made by the stochastic OT learner over ten runs on the counterexample (15) minus the number of errors made in that same run by the deterministic learner; see table 6 in appendix E for simulation results. Comparison with the function $5 \times n^2$ plotted by the dashed line[23] shows that the number of additional errors due to the stochastic component grows slowly with the number of constraints, so that the learner makes less than 200 additional errors in the case with ten constraints.[24]

(28)



These simulation results are not surprising. In fact, assume that the stochastic OT learner does not promote "too much", namely adopts a promotion component (12a) corresponding to a calibration constant c which is not "too large", namely strictly smaller than 1. Whenever trained on data consistent with an OT grammar, the number of errors is bounded by (29), which depends on the number of constraints and on the calibration constant c (Magri 2015c).[25]

[22] See footnote 25 for discussion of the dependence of the bound (27) on the constant Δ, which was interpreted in footnote 11 as the size of the stochastic component.

[23] The function $5 \times n^2$ has been chosen here because the exact version of the OT error bound (29) is $\frac{1+2\Delta}{2(1-c)} n(n-1)$ (Magri 2015c), which boils down to $5n(n-1) \sim 5n^2$ when $c = 1/2$ and $\Delta = 2$, as in the simulations reported here.

[24] In the case of the HG learner (with either reweighing rule), the difference between the final weights of constraint C_1 entertained at the end of the stochastic and deterministic runs grows exponentially with the number of constraints, as plotted in (26b). In the case of the OT learner instead, the final ranking values of C_1 in the deterministic and stochastic runs are essentially the same, independently of the number of constraints: the difference is at most 0.5 for 5 constraints and at most 0.7 for 10 constraints, as shown in table 6 of appendix E.

[25] As anticipated in footnote 5, the number of additional errors due to the stochastic implementation is expected to grow not only with the number of constraints but also with the "size" of the stochastic component. As anticipated in footnote 11, the size of the stochastic component is the constant Δ, as the stochastic values $\epsilon_1, \epsilon_2, \ldots$ added to the current weights or ranking values are sampled between $-\Delta$ and $+\Delta$. Indeed, the larger Δ, the larger the effect of the stochastic component

$$\text{number of errors} \lessapprox \underbrace{\frac{\big(\text{number of constraints}\big)^2}{1-c}}_{\text{(a)}} + \underbrace{\frac{\Delta \times \big(\text{number of constraints}\big)^2}{1-c}}_{\text{(b)}} \tag{29}$$

This error-bound (29) is the sum of two terms. The first term (29a) coincides with the error-bound (21) for the deterministic implementation. The second term (29b) thus expresses the number of additional errors due to the stochastic implementation. This additional term (29b) grows slowly (quadratically) with the number of constraints. Furthermore, the bound requires no assumptions on the constraints. It thus settles question (6c) concerning constraint-independent guarantees that the OT learner tolerates the stochastic component.

5.4. **Summary**

Section 4 has compared the OT and HG *deterministic* implementations of error-driven learning. This section has extended the comparison to the *stochastic* implementations. It has focused on the number of additional errors made by the learner because of the stochastic component. The error-bound for the OT stochastic learner provides constraint-independent guarantees that it tolerates the stochastic component. The constraint-independent bound currently available for the HG stochastic learner instead does *not* guarantee stochastic tolerance, because of its dependence on the margin of the data. Indeed, the HG learner does not tolerate the stochastic implementation constraint-independently: with just ten constraints, the stochastic HG learner can make almost one million and a half additional errors.

6. Noise robustness

This section discusses some counterexamples where the number of additional errors made by the HG learner in response to a small faulty update triggered by a single noisy piece of data grows fast (exponentially) with the number of constraints, showing that noise-robustness does not hold constraint-independently, contrary to the case of the OT learner.

6.1. **Counterexample**

Faulty updates triggered by corrupted data disrupt the current weights or ranking values and thus require additional updates by pristine data to recover. The learner is robust to noise provided the number of updates it takes to recover is small. To estimate noise robustness, we need to inflict the largest damage with each single corrupted piece of data. For concreteness, let me focus on the HG learner with the truncated reweighing rule (the reasoning extends straightforwardly to the original Perceptron reweighing rule). Consider again the counterexample (15) used in the two preceding sections 4 and 5. Table (30) reports the final weights learned at the end of a run on the counterexample corresponding to a number n of constraints between $n = 5$ and $n = 10$.

(30)

	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8	θ_9	θ_{10}
$n=5$	26	9	3	1	0					
$n=6$	71	26	9	3	1	0				
$n=7$	175	65	24	9	3	1	0			
$n=8$	424	168	65	24	9	3	1	0		
$n=9$	1026	417	164	64	24	9	3	1	0	
$n=10$	2510	1029	416	166	64	24	9	3	1	0

In section 4, we have focused on constraint C_1 and the fact that its final weight grows exponentially with the number of constraints. We now focus instead on the last two constraints C_{n-1} and C_n. Table (30) shows that their weights are always apart by just 1 at convergence. We can take advantage of this fact as follows. Let's train the learner on pristine data sampled from the counterexample (15) until it convergences to the final weights (30). At that point, we feed the learner just one corrupted piece of data which looks as in (31): the violation difference corresponding to constraint C_1 is -1; the violation difference corresponding to the penultimate constraint C_{n-1} is $+1$; the violation difference corresponding to the last constraint C_n is $+2$; all other violation differences are null.

(31)

C_1	C_2	...	C_{n-2}	C_{n-1}	C_n
-1	0	—	0	$+1$	$+2$

on the current weights or ranking values. The bounds (27b) and (29b) say that the number of additional errors due to the stochastic implementation grows slowly (linearly) with the size Δ of the stochastic component in both the HG and the OT implementations.

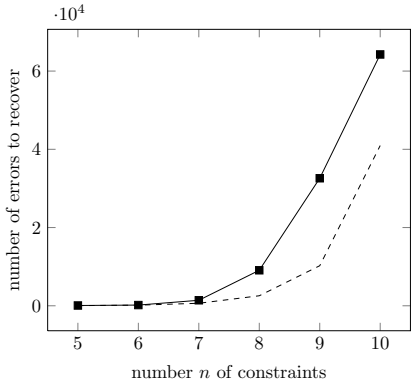
Since the violation difference corresponding to constraint C_1 is negative in (31) and the weight of C_1 is very large in (30), this corrupted piece of data (31) is inconsistent with the weights (30), which are therefore updated to the weights in (32), where I have boldfaced the updated weights. Because of careful choice of the violation difference corresponding to the last two constraints C_{n-1} and C_n, they end up with the same weight 2 in (32). Consistency with the pristine training data (15) instead requires the weight of C_{n-1} to be larger than the weight of C_n. In order to recover the consistent weight configuration, the bottom row of the counterexample (15) has to trigger an update. That update will promote C_{n-1} by 1, to the weight 3. But constraints C_{n-2} and C_{n-1} will then have exactly the same weight 3. Consistency with the pristine training data (15) instead requires the weight of C_{n-2} to be larger than the weight of C_{n-1}. As this reasoning can be repeated backward for all the constraints, we expect a domino effect.

(32)

	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8	θ_9	θ_{10}
$n=5$	**25**	9	3	**2**	**2**					
$n=6$	**70**	26	9	3	**2**	**2**				
$n=7$	**174**	65	24	9	3	**2**	**2**			
$n=8$	**423**	168	65	24	9	3	**2**	**2**		
$n=9$	**1025**	417	164	64	24	9	3	**2**	**2**	
$n=10$	**2509**	1029	416	166	64	24	9	3	**2**	**2**

Indeed, the solid line in (33) plots the largest number of errors made by the HG learner (with the truncated Perceptron reweighing rule) over ten runs on the counterexample (15) starting from the weights (32); see table 9 in appendix E for simulation results. Comparison with the function $40 \times 4^{n-5}$ plotted by the dashed line shows that the number of updates needed to recover from just a single small faulty update grows fast (exponentially) with the number n of constraints. So fast that the learner requires over 64,000 updates in the case with just ten constraints, despite the weights (30) and (32) before and after the faulty update being so similar.

(33)



In conclusion, this counterexample shows that constraint-independent guarantees of noise-robustness for the HG error-driven learner are impossible, as a single faulty update can require a large number of updates to recover. Any such guarantees will have to be restricted by specific assumptions on the constraints.

6.2. **Discussion/I: the margin, once again**

Suppose that the HG learner is trained on a (possibly infinite) sequence of *pristine* training data generated by some target HG grammar interspersed with a finite number of data *corrupted* by transmission noise or production errors. No assumptions are made on the corrupted data, apart from there being only a finite number of them. The number of errors made by the HG Perceptron learner in this noisy learning setting is bounded by (34) (Freund and Schapire 1999, building on Klasner and Simon 1995; later developments such as Shalev-Shwartz and Singer 2005 and Mohri and Rostamizadeh 2013 do not alter the shape of the bound). The precise definition of the quantity at the numerator of (34b) is somewhat involved and therefore omitted (for an accessible description, see Magri 2015a). Suffice it to say that this quantity is null when there are no corrupted training data and grows with the number of corrupted data. The error bound extends to the *truncated* Perceptron reweighing rule (11), only with the margin of the pristine data at the numerator replaced by the margin of the pristine plus the dummy data (Magri 2015a).

$$\text{(34)}\quad \text{number of errors} \lessapprox \underbrace{\frac{\text{number of constraints}}{\big(\text{margin of the pristine data}\big)^2}}_{\text{(a)}} + \underbrace{\frac{\text{a certain quantity which depends on the corrupted data}}{\big(\text{margin of the pristine data}\big)^2}}_{\text{(b)}}$$

This error-bound (34) is the sum of two terms. The first term (34a) coincides with the error-bound (17) for the (deterministic) HG learner in the noise-free setting. The second term (34b) thus quantifies the number of additional errors due to the corrupted data. This second term depends on the number (and the shape) of the corrupted data through the numerator. It also depends on the margin of the pristine data through the denominator. As discussed in subsection 4.3, the margin depends on the number of constraints. In the case of the counterexample (15), its squared inverse grows fast (exponentially) with the number of constraints, as plotted in (18). The bound (34b) on the number of additional errors due to the noisy data thus grows fast and does not settle question (6d) concerning constraint-independent noise robustness. That question is answered negatively by the counterexample described in subsection 6.1, which shows that constraint-independent guarantees for noise robustness are impossible in HG.

6.3. **Discussion/II: comparison with OT**

In the case of ten constraints, nine updates suffice for the OT learner to recover from the faulty update described in subsection 6.1. These simulation results are not surprising. In fact, consider an OT learner which does not promote "too much", namely which adopts a promotion component (12a) corresponding to a calibration constant c which is not "too large", namely strictly smaller than 1. Suppose that the learner is trained on a (possibly infinite) sequence of data consistent with a certain OT grammar, interspersed with a finite number of corrupted data. The learner admits the error-bound (35), which depends on the number of constraints, the calibration constant used in the definition of the promotion amount, and the number of corrupted training data (Magri 2015c).

$$\text{(35)}\quad \text{error-bound} \lessapprox \underbrace{\frac{(\text{number of constraints})^2}{1-c}}_{\text{(a)}} + \underbrace{\frac{\text{number of constraints}}{1-c} \times \text{number of corrupted data}}_{\text{(b)}}$$

This error-bound (35) is the sum of two terms. The first term (35a) coincides with the error-bound (21) for the noise-free setting. The second term (35b) thus expresses the number of additional errors due to the corrupted training data. The latter term (35b) is in turn the product between the number of corrupted data times a factor which thus quantifies the amount of disruption caused by each corrupted piece of data. This additional term (35b) grows slowly (linearly) with the number of constraints (as well as with the number of corrupted data). Furthermore, this bound requires no assumptions on the constraints. It thus settles question (6d) concerning constraint-independent guarantees that the OT learner is robust to noise.

6.4. **Summary**

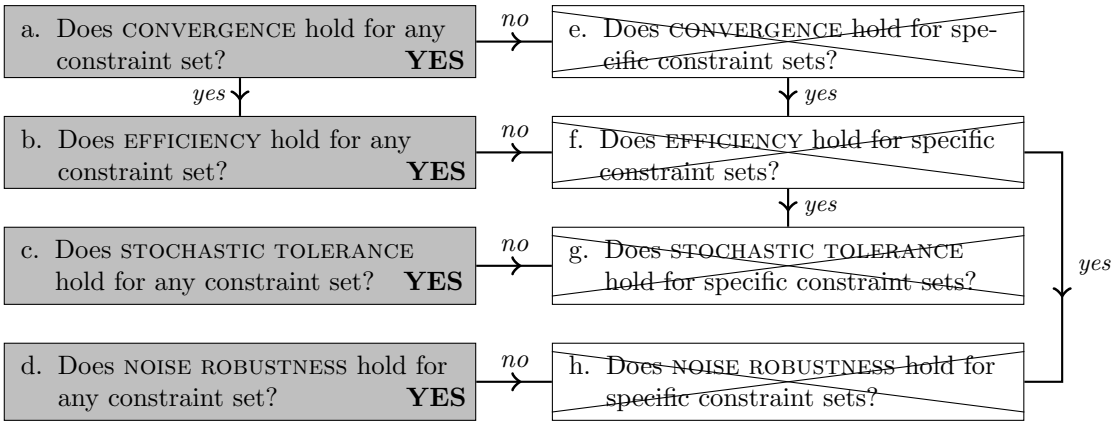
Sections 4 and 5 have compared the OT and HG implementations of error-driven learning in the *noise-free* setting. This section has extended the comparison to the *noisy* learning setting. It has focused on the number of additional errors made by the learner to recover from the faulty updates triggered by corrupted training data. The OT learner comes with a bound on the number of additional errors which provides constraint-independent guarantees on its noise robustness. The constraint-independent bound currently available for the HG learner instead does not guarantee noise robustness, because of its dependence on the margin of the data. Indeed, the HG learner is not noise robust constraint-independently: with just ten constraints, the HG learner can make more than 64,000 additional updates to recover from a faulty update triggered by a single piece of corrupted data.

7. Conclusions

The theories of OT and HG error-driven learning which emerge from the results presented in this paper are synopsized in (36) and (37), using the scheme (6). Various bounds for the OT learner have been recalled which provide guarantees of *convergence* (is the number of errors finite?), *efficiency* (is the number of errors furthermore small?), *stochastic tolerance* (does the number of errors remain small in the stochastic implementation?), and *noise robustness* (does the number of errors remain small in the noisy setting?). These bounds follow from formal properties of the OT mode of constraint interaction and thus hold for any constraints. They thus provide a positive answer to the four constraint-independent questions

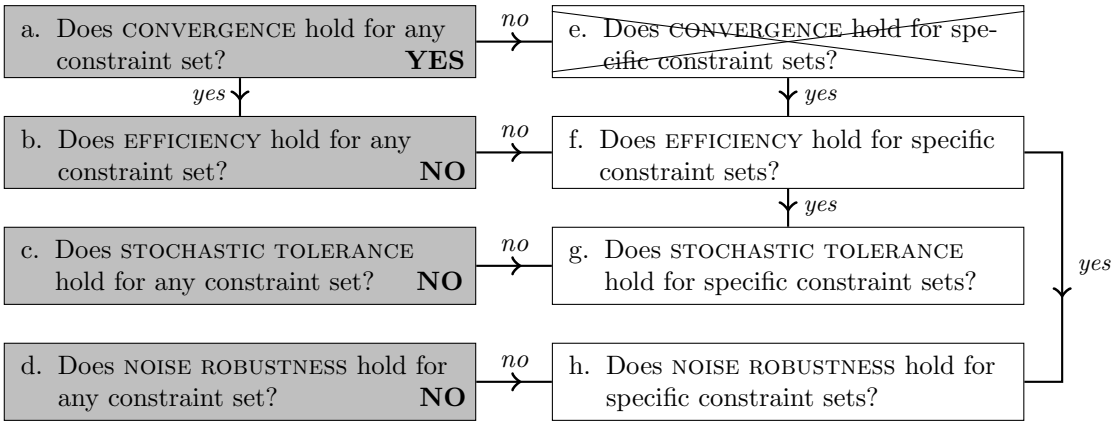
(36a)-(36d). The four constraint-dependent questions (36e)-(36h) become superfluous. The theory of OT error-driven learning has thus been successfully completed from a constraint-independent perspective.

(36) *OT theory of error-driven learning:*



Various bounds for the Perceptron HG learner (and their extensions to the truncated variant which ensures nonnegative weights) have also been recalled. They guarantee convergence irrespectively of the constraints, and thus provide a positive answer to the constraint-independent question (37a). Unfortunately, these bounds all depend on the margin of the data, a measure of the slackness in the choice of consistent weights. In subsection 4.2, I have described a class of cases where the margin decreases fast with the number of constraints and the available bounds thus grow fast and provide no guarantees. Indeed, the HG learner has been shown not to converge efficiently in these cases, not to tolerate the stochastic implementation, and not to be robust to noise. Constraint-independent guarantees are thus impossible for the HG learner, yielding a negative answer to the three questions (37b)-(37d).

(37) *HG theory of error-driven learning:*



Together, (36) and (37) show that the theory of error-driven learning in OT and HG has a completely different nature (as foreshadowed in Prince and Smolensky 2004, section 10.2.2).

The difference between OT and HG just brought out is best appreciated against the backdrop of a recent debate in computational phonology. Pater (2008) provides an elegant counterexample against Boersma's (1998) GLA implementation of OT error-driven learning. He then diagnoses that "one reason for the failure [of the GLA on his counterexample], and potentially for its failure to provably converge in general, is its use of [...] OT as a model of grammar." He then notes that "the GLA closely resembles Rosenblatt's (1958) Perceptron", which instead uses the HG model of grammar. And he thus concludes advocating the replacement of OT with HG and the adoption of the Perceptron update rule for HG error-driven learning. Pater (2009, p. 1021) indeed argues that "one broad argument for weighted constraints [...] is that weighted constraints are compatible with existing well-understood algorithms for learning variable outcomes and for learning gradually [...]." The results synopsized in (36) and (37) turn this assessment on its head.

To start, the problem with Pater's counterexample is not germane to the OT mode of constraint interaction. Rather, it is only due to a specific GLA's implementation detail, namely the GLA's choice of a fixed

promotion amount. As recalled in subsection 4.4, the problem evaporates through the adoption of a promotion amount which is carefully calibrated on the number of winner-preferring constraints, as prescribed by the promotion rule (12a). Indeed, convergence, efficiency, stochastic tolerance, and noise robustness are guaranteed for such an OT learner. And these guarantees hold independently of the phonological content of the constraints, circumventing the contentious issue of the characterization of phonologically plausible constraints.

The situation turns out to be very different for the HG learner. We have seen that the HG learner can make an astronomical number of errors: for instance, it makes more than 6,500,000 errors in the counterexample (15) with just ten constraints. Hence, no constraint-independent efficiency guarantees are possible. Any HG efficiency guarantees will have to be restricted through proper assumptions on the constraints. The difficulty in developing such constraint-dependent analyses is that the behavior of the HG learner can change abruptly in the face of even very small modifications of the training data. Indeed, we have seen that the number of errors drops to just 200 in the variant of the counterexample (15) considered in (23), despite the fact that the two test cases differ only minimally (only two constraint violation differences have been changed by no more than 2). Finally, the behavior of the HG learner can change abruptly in the face of even small modifications of the the current weights. Indeed, we have seen that a single, small update of the current weights (which changes only three weights by less than 2) has a dramatic effect and requires the learner to perform over 64,000 additional updates to recover. These abrupt discontinuities in the behavior of the HG learner in response to only small modifications of the data or the weights overturn Pater's assessment of the relative merits of OT and HG from the perspective of error-driven learning.

This conclusion in turn has implications for the connection between constraint-based phonology and the computational literature. There has been a spur of interest in learning with rankings in the recent machine learning literature (under the heading of *preference learning*; Fürnkranz and Hüllermeier 2010). Because of its constraint-independent nature, the OT theory of error-driven learning is independent of the phonological substance and might thus benefit from the methods and results developed within the computational literature. The HG theory must instead rely on substantive assumptions on the phonological content of specific constraints. In other words, the HG theory must focus on the implications for efficiency and robustness of subtle assumptions on phonologically plausible constraints. The machine learning literature has plausibly little to contribute to that enterprise. The HG theory of error-driven learning is thus unlikely to benefit from the analyses of general-purpose "existing well-understood algorithms" from the machine learning literature, contrary to Pater's suggestion (see for instance subsection 4.9 above).

Of course, the choice between the OT and HG implementations of constraint-based phonology is an empirical issue which should in the end be decided by typological considerations. Yet, recent research has started to compare the two frameworks from the perspective of their acquisition and learnability implications (Riggle 2009; Bane et al. 2010; Magri 2013b; Jesney and Tessier 2011). This paper contributes to this line of research, through a re-evaluation of the relative merits of OT and HG from the perspective of error-driven learnability. The relevance of this re-evaluation can be concretely illustrated through the following example. Levelt et al. (2000) report that Dutch learning children go through acquisition stages where they allow a single marked structure while doubly marked structures lag behind. For instance, they allow syllables with complex codas and allow syllables with complex onsets but do not allow syllables where these two markedness violations gang-up. Do these acquisition data show that child language displays gang-up effects and therefore requires HG over OT? The answer is delicate. In fact, although OT does not capture gang-up effects by design, Jäger and Rosenbach (2006) show that some of these effects can be modeled in OT through the addition of a stochastic component. Building on this observation, Jarosz (2010) shows that stochastic OT is able to model the gang-up effects reported in Levelt's child data (with standard constraints for syllable types) and that an OT error-driven learner indeed walks through intermediate stages which correspond to the attested gang-up effects. Which account of these gang-up effects is thus to be preferred, the stochastic OT account or the HG one? This is clearly a case where descriptive adequacy falls short. The computational results reported in this paper might help in this case. In fact, while the HG learner is not efficient from a constraint-independent perspective, the stochastic implementation does not affect the efficiency of the OT learner. These computational results might thus provide a starting point for a learnability-based argument which fills the gap left by the insufficiency of sheer phonological data.

References

Anderson, James A., and Edward Rosenfeld. 1988. *Neurocomputing: Foundations of research*. Cambridge: MIT Press.

Bane, Max, Jason Riggle, and Morgan Sonderegger. 2010. The VC dimension of constraint-based grammars. *Lingua* 120.5:1194–1208.

Bíró, Tamás Sándor. 2006. Finding the right words: Implementing Optimality Theory with Simulated Annealing. Doctoral Dissertation, University of Groningen. Available as ROA-896.

Block, H. D. 1962. The perceptron: A model of brain functioning. *Review of Modern Physics* 34:123–135. Reprinted in Anderson and Rosenfeld (1988).

Boersma, Paul. 1997. How we learn variation, optionality and probability. In *Proceedings of the Institute of Phonetic Sciences (IFA) 21*, ed. Rob van Son, 43–58. University of Amsterdam: Institute of Phonetic Sciences.

Boersma, Paul. 1998. Functional phonology. Doctoral Dissertation, University of Amsterdam, The Netherlands. The Hague: Holland Academic Graphics.

Boersma, Paul. 2009. Some correct error-driven versions of the constraint demotion algorithm. *Linguistic Inquiry* 40:667–686.

Boersma, Paul, and Bruce Hayes. 2001. Empirical tests for the Gradual Learning Algorithm. *Linguistic Inquiry* 32:45–86.

Boersma, Paul, and Jan-Willem van Leussen. 2014. Fast evaluation and learning in multi-level parallel constraint grammars. University of Amsterdam.

Boersma, Paul, and Joe Pater. to appear. Convergence properties of a gradual learning algorithm for Harmonic Grammar. In *Harmonic Grammar and Harmonic Serialism*, ed. John McCarthy and Joe Pater. London: Equinox Press.

Cesa-Bianchi, Nicolò, and Gábor Lugosi. 2006. *Prediction, learning, and games*. Cambridge University Press.

Chomsky, Noam. 1965. *Aspects of the theory of syntax*. MIT Press.

Coetzee, Andries W., and Shigeto Kawahara. 2013. Frequency biases in phonological variation. *Natural Language and Linguistic Theory* 31:47–89.

Coetzee, Andries W., and Joe Pater. 2008. Weighted constraints and gradient restrictions on place co-occurrence in Muna and Arabic. *Natural Language and Linguistic Theory* 26:289–337.

Coetzee, Andries W., and Joe Pater. 2011. The place of variation in phonological theory. In *Handbook of phonological theory*, ed. John Goldsmith, Jason Riggle, and Alan Yu, 401–434. Cambridge: Blackwell.

Collins, Michael. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, ed. Jan Haji and Yuji Matsumoto, 1–8. Association for Computational Linguistics.

Cristianini, Nello, and John Shawe-Taylor. 2000. *An introduction to Support Vector Machines and other Kernel-based methods*. Cambridge University Press.

Eisner, Jason. 2000. Easy and hard constraint ranking in Optimality Theory. In *Finite-State Phonology: ACL Special Interest Group in Computational Phonology (SIGPHON) 5*, ed. Jason Eisner, Lauri Karttunen, and Alain Thériault, 22–33. San Francisco, CA: Morgan Kaufmann Publishers / ACL.

Frank, Robert, and Shyam Kapur. 1996. On the use of triggers in parameter setting. *Linguistic Inquiry* 27:623–660.

Freund, Yoav, and Robert E. Schapire. 1999. Large margin classification using the Perceptron algorithm. *Machine Learning* 37:277–296.

Fürnkranz, Johannes, and Eyke Hüllermeier. 2010. *Preference learning*. Springer.

Gibson, Edward, and Kenneth Wexler. 1994. Triggers. *Linguistic Inquiry* 25:407–454.

Hayes, Bruce. 2004. Phonological acquisition in Optimality Theory: The early stages. In *Constraints in phonological acquisition*, ed. René Kager, Joe Pater, and Wim Zonneveld, 158–203. Cambridge: Cambridge University Press.

Heinz, Jeffrey. 2011. Computational phonology. part i: Foundations. *Linguistic Compass* 5.4:140–152.

Jarosz, Gaja. 2010. Implicational markedness and frequency in constraint-based computational models of phonological learning. *Journal of Child Language* 37:565–606.

Jarosz, Gaja. 2013. Learning with hidden structure in Optimality Theory and Harmonic Grammar: Beyond Robust Interpretative Parsing. *Phonology* 30:27–71.

Jesney, Karen, and Anne-Michelle Tessier. 2011. Biases in Harmonic Grammar: the road to restrictive learning. *Natural Language and Linguistic Theory* 29:251–290.

Jäger, Gerhard, and Anette Rosenbach. 2006. The winner takes it all–almost. Cumulativity in grammatical variation. *Linguistics* 44:937–971.

Keller, Frank. 2000. Gradience in grammar. Experimental and computational aspects of degrees of grammaticality. Doctoral Dissertation, University of Edinburgh, England.

Kivinen, Jyrki. 2003. Online learning of linear classifiers. In *Advanced lectures on machine learning (lnai 2600)*, ed. S. Mendelson and A.J. Smola, 235–257. Berlin Heidelberg: Springer-Verla.

Klasner, Norbert, and Hans-Ulrich Simon. 1995. From noise-free to noise-tolerant and from on-line to batch learning. In *Computational Learning Theory (COLT) 8*, ed. Wolfgang Maass, 250–257. ACM.

Legendre, Gèraldine, Yoshiro Miyata, and Paul Smolensky. 1998a. Harmonic Grammar: A formal multi-level connectionist theory of linguistic well-formedness: An application. In *Annual conference of the Cognitive Science Society 12*, ed. Morton Ann Gernsbacher and Sharon J. Derry, 884–891. Mahwah, New Jersey: Lawrence Erlbaum Associates.

Legendre, Géraldine, Yoshiro Miyata, and Paul Smolensky. 1998b. Harmonic Grammar: A formal multi-level connectionist theory of linguistic well-formedness: Theoretical foundations. In *Annual conference of the Cognitive Science Society 12*, ed. Morton Ann Gernsbacher and Sharon J. Derry, 388–395. Mahwah, NJ: Lawrence Erlbaum.

Legendre, Gèraldine, Antonella Sorace, and Paul Smolensky. 2006. The optimality theory/harmonic grammar connection. In *The harminic mind*, ed. Paul Smolensky and Gèraldine Legendre, 903–966. Cambridge, MA: MIT Press.

Levelt, Clara C., Niels O. Schiller, and Willem J. Levelt. 2000. The acquisition of syllable types. *Language Acquisition* 8(3):237–264.

Magri, Giorgio. 2012a. Constraint promotion: not only convergent, but also efficient. In *Proceedings of the 48th annual conference of the Chicago Linguistics Society*.

Magri, Giorgio. 2012b. Convergence of error-driven ranking algorithms. *Phonology* 29:213–269.

Magri, Giorgio. 2013a. The complexity of learning in OT and its implications for the acquisition of phonotactics. *Linguistic Inquiry* 44.3:433–468.

Magri, Giorgio. 2013b. HG has no computational advantages over OT: towards a new toolkit for computational OT. *Linguistic Inquiry* 44.4:569–609.

Magri, Giorgio. 2015a. How to keep the HG weights non-negative: the truncated perceptron reweighing rule. *Journal of Language Modeling* .

Magri, Giorgio. 2015b. Idempotency in Optimality Theory. Manuscript.

Magri, Giorgio. 2015c. Noise robustness and stochastic tolerance of OT error-driven ranking algorithms. *Journal of Logic and Computation* .

Magri, Giorgio, and Benjamin Storme. forth. A closer look at Boersma and Hayes' Ilokano metathesis test case. In *Proceedings of the 49th annual conference of the Chicago Linguistics Society*.

Minsky, Marvin, and Seymour Papert. 1969. *Perceptrons: An introduction to Computational Geometry*. MIT Press.

Mohri, Mehryar, and Afshin Rostamizadeh. 2013. Perceptron mistake bounds. ArXiv:1305.0208.

Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar. 2012. *Foundations of machine learning*. Cambridge, MA: MIT Press.

Novikoff, Albert B. J. 1962. On convergence proofs on Perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume XII, 615–622.

Pater, Joe. 2008. Gradual learning and convergence. *Linguistic Inquiry* 39:334–345.

Pater, Joe. 2009. Weighted constraints in Generative Linguistics. *Cognitive Science* 33:999–1035.

Prince, Alan. 2002. Entailed ranking arguments. Ms., Rutgers University, New Brunswick, NJ. Rutgers Optimality Archive, ROA 500. Available at http://www.roa.rutgers.edu.

Prince, Alan, and Paul Smolensky. 2004. *Optimality Theory: Constraint interaction in generative grammar*. Oxford: Blackwell. As Technical Report CU-CS-696-93, Department of Computer Science, University of Colorado at Boulder, and Technical Report TR-2, Rutgers Center for Cognitive Science, Rutgers University, New Brunswick, NJ, April 1993. Also available as ROA 537 version.

Prince, Alan, and Bruce Tesar. 2004. Learning phonotactic distributions. In *Constraints in phonological acquisition*, ed. R. Kager, J. Pater, and W. Zonneveld, 245–291. Cambridge University Press.

Riggle, Jason. 2009. The complexity of ranking hypotheses in Optimality Theory. *Computational Linguistics* 35(1):47–59.

Rosenblatt, Frank. 1958. The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* 65:386–408.

Rosenblatt, Frank. 1962. *Principles of Neurodynamics*. New York: Spartan.

Shalev-Shwartz, Shai, and Yoram Singer. 2005. A new perspective on an old perceptron algorithm. In *Conference on Computational Learning Theory (COLT) 18*, ed. Peter Auer and Ron Meir, 264–278. Lecture Notes in Computer Science, Springer.

Smolensky, Paul, and Gèraldine Legendre. 2006. *The harmonic mind*. Cambridge, MA: MIT Press.

Staubs, Robert, Michael Becker, Christopher Potts, Patrick Pratt, John J. McCarthy, and Joe Pater. 2010. OT-Help 2.0. Software package. Amherst, MA: University of Massachusetts Amherst.
Tesar, Bruce. 2004. Using inconsistency detection to overcome structural ambiguity. *Linguistic Inquiry* 35.2:219–253.
Tesar, Bruce. 2013. *Output-driven phonology: Theory and learning*. Cambridge Studies in Linguistics.
Tesar, Bruce, and Paul Smolensky. 1998. Learnability in Optimality Theory. *Linguistic Inquiry* 29:229–268.
Tesar, Bruce, and Paul Smolensky. 2000. *Learnability in Optimality Theory*. Cambridge, MA: MIT Press.
Vapnik, Vladimir N. 1998. *Statistical learning theory*. John Wiley and sons.
Wexler, Kenneth, and Peter W. Culicover. 1980. *Formal principles of language acquisition*. Cambridge, MA: MIT Press.

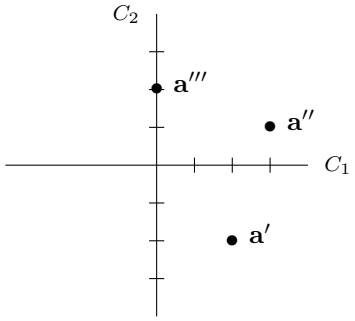
Appendix A. Margin

At each iteration, the error-driven learner chews a piece of data which consists of a certain winner candidate y and a certain loser candidate $\not{y}$ for a certain underlying form x. Let me denote by a_k the corresponding violation difference for constraint C_k, as in (38). That is the difference between the number $C_k(x, \not{y})$ of violations assigned by constraint C_k to the loser mapping minus the number $C_k(x, y)$ of violations assigned to the winner-mapping. Collect the corresponding violation differences corresponding to all constraints $C_1, C_2, \dots$ into a vector $\mathbf{a} = (a_1, a_2, \dots)$, called an *elementary weighting condition* (EWC), by analogy with Prince's (2002) *elementary ranking conditions* in Optimality Theory.

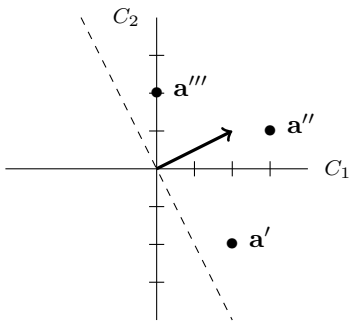
(38) $a_k = C_k(x, \not{y}) - C_k(x, y)$

Suppose there are only $n = 2$ constraints C_1 and C_2. A generic EWC thus has the shape $\mathbf{a} = (a_1, a_2)$: it consists of the differences a_1 and a_2 between the number of violations assigned to the loser minus the number of violations assigned to the winner by the two constraints C_1 and C_2, respectively. An EWC can thus be represented with a point in the cartesian plane, through the convention that the horizontal axis corresponds to constraint C_1 and the vertical axis corresponds to constraint C_2. To illustrate, the three EWCs $\mathbf{a}' = (2, -2)$, $\mathbf{a}'' = (3, 1)$, and $\mathbf{a}''' = (0, 2)$ can be represented as in (39).

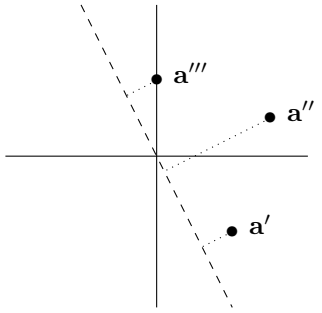
(39)



With only $n = 2$ constraints, a generic weight vector has the shape $\boldsymbol{\theta} = (\theta_1, \theta_2)$: it consists of the weights θ_1 and θ_2 of the two constraints C_1 and C_2. It can be represented through an arrow which starts at the origin and points at the point with coordinates θ_1 and θ_2. To illustrate, the weight vector $\boldsymbol{\theta} = (2, 1)$ is represented by the arrow in (40). The *decision line* corresponding to a weight vector is the line through the origin which is perpendicular to the arrow which represents the weight vector. To illustrate, the decision line corresponding to the weight vector $\boldsymbol{\theta} = (2, 1)$ is represented by the dashed line in (40). The decision line splits the plane into two regions, one of which contains the arrow. The HG-consistency condition (8) between an underlying/winner/loser form triplet $(x, y, \not{y})$ and a weight vector says that the EWC corresponding to that triplet lies in the half-plane which contains the arrow corresponding to that weight vector. To illustrate, (40) shows a weight vector consistent with each of the three EWCs $\mathbf{a}', \mathbf{a}'', \mathbf{a}'''$, as they lie in the half-plane containing the arrow.

(40) 

The *distance* of an EWC from the decision line is the length of the segment which starts at the EWC and falls perpendicularly on the decision line, represented by the dotted segments in (41). This distance can be interpreted as a measure of the "degree of consistency" of the EWC with (the decision line corresponding to) the weight vector. Thus, although the weight vector plotted in (41) is consistent with both EWCs $\mathbf{a}'$ and $\mathbf{a}''$, the former EWC has a smaller degree of consistency than the latter. Indeed, a small perturbation of the weights slightly rotates the decision line and might affect consistency with the closer EWC $\mathbf{a}'$ but not with the EWC $\mathbf{a}''$. Since we are interested in worst-case analyses, we focus on the most "dangerous" piece of training data, namely the underlying/winner/loser form triplet whose EWC has the smallest degree of consistency because it is closest to the decision line. The distance of that EWC from the decision line is called the *margin* of the training data with respect to (the decision line corresponding to) the weight vector $\boldsymbol{\theta}$. To illustrate, the margin of the training data corresponding to the three EWCs $\mathbf{a}', \mathbf{a}'', \mathbf{a}'''$ relative to the decision line represented by the dashed line in (41) is the distance of either EWC $\mathbf{a}'$ or $\mathbf{a}'''$.

(41) 

Different weight vectors induce different decision lines that in turn differ because of their distances from the training EWCs. Among all weight vectors consistent with the data set, we thus consider a weight vector $\hat{\boldsymbol{\theta}}$ whose decision line achieves the largest distance from the closest EWC, namely whose margin is at least as large as the margin relative to any other weight vector $\boldsymbol{\theta}$. The margin of any such *optimal* weight vector $\hat{\boldsymbol{\theta}}$ is called the *margin* of the training data. As it is clear from this geometric definition, all optimal weight vectors correspond to the same decision line, which is therefore unique. The extension from $n = 2$ to an arbitrary number n of constraints is conceptually straightforward.

Appendix B. The margin of the counterexample

Consider again the counterexample constructed in subsection 4.2. Let $\mathbf{A}_n$ be the EWC matrix in (15a). Let $\| \cdot \|$ denote the *Euclidean norm*, defined by $\|\mathbf{v}\|^2 = \sum_{i=1}^n v_i^2$. Let $\mathbf{A}_n\boldsymbol{\theta}$ denote the row-by-column product between the matrix $\mathbf{A}_n$ and the (column) weight vector $\boldsymbol{\theta}$. For any two vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$, the inequality $\mathbf{v} \geq \mathbf{w}$ (and $\mathbf{v} > \mathbf{w}$) means that each component of $\mathbf{v}$ is at least as large as (is strictly larger than, respectively) the corresponding component of $\mathbf{w}$. Vapnik's (1998) Theorem 10.2 ensures that the squared inverse of the margin of the training data corresponding to n constraints coincides with the (unique) solution of the optimization problem (42) in the unknown $\boldsymbol{\theta} \in \mathbb{R}^n$.

(42) minimize: $\|\boldsymbol{\theta}\|^2$
subject to: $\mathbf{A}_n\boldsymbol{\theta} \geq 1$

As explained in subsection 4.3, define the *dummy data* as the set of all underlying/winner/loser form triplets whose constraint violation differences are all null apart from one, which is equal to +1 (Magri 2015a). In the case of n constraints, the corresponding matrix of EWCs is thus the identity matrix $\mathbb{I}_n$ (namely the matrix with n columns and n rows, whose entires are all equal to 0 but for the diagonal entries which are all equal to 1). And the squared inverse margin of the training data plus the dummy

data corresponding to n constraints coincides with the (unique) solution of the optimization problem (43) in the unknown $\boldsymbol{\theta} \in \mathbb{R}^n$.

$$\begin{array}{lll} (43) & \text{minimize:} & \|\boldsymbol{\theta}\|^2 \\ & \text{subject to:} & \mathbf{A}_n\boldsymbol{\theta} \geq 1 \\ & & \mathbb{I}_n\boldsymbol{\theta} \geq 1 \end{array}$$

As (42) and (43) are convex quadratic programs, they can be solved with standard optimization software, such as the `Matlab` optimization toolbox. The results for $n = 5, 6, 7, 8, 9, 10$ are plotted in (18).

Appendix C. The HG learner mimics the OT learner on the counterexample

This appendix explains intuitively the claim made in subsection 4.7 that the HG learner with the original Perceptron reweighing rule (and properly chosen step sizes) mimics the OT learner with the calibration constant c set equal to $c = 1$ on the counterexample (15). To reason concretely, consider the case with 5 constraints and suppose that the learner is currently being trained on the penultimate row of the corresponding matrix (15b), repeated in (44).

$$(44) \qquad \begin{array}{c} \begin{array}{ccccc} C_1 & C_2 & C_3 & C_4 & C_5 \end{array} \\ \left[\begin{array}{ccccc} & & +1 & -2 & +1 \end{array}\right] \end{array}$$

Suppose that this piece of data is inconsistent with the current ranking vector $\boldsymbol{\theta}$ and thus prompts the OT learner to update. This means that the ranking values θ_3 and θ_5 of the two winner-preferring constraints C_3 and C_5 are each smaller than the ranking value θ_4 of the loser-preferring constraint C_4. The sum $\theta_3+\theta_5$ of the ranking values of the two winner-preferring constraints is thus smaller than twice the ranking value θ_4 of the loser-preferring constraint, namely $\theta_3 + \theta_5 < 2\theta_4$. Since the size of the violation difference of the loser-preferring constraint C_4 is indeed 2 in (44), the latter inequality says that the average of the violation differences (44) weighted by the weights $\boldsymbol{\theta}$ is negative. In other words, the vector $\boldsymbol{\theta}$ is not HG-consistent with (44) so that the HG learner would update as well. This conclusion holds in complete generality: since the size of the negative entries in the counterexample (15) is always equal to the number of positive entries, an update by the OT learner entails an update by the HG learner (see Magri (2013b, section 4) for discussion of the general case). This conclusion crucially requires the size of the negative entries to be equal to the number of positive entries. For instance, consider the variant in (45), with -2 replaced by -3.

$$(45) \qquad \begin{array}{c} \begin{array}{ccccc} C_1 & C_2 & C_3 & C_4 & C_5 \end{array} \\ \left[\begin{array}{ccccc} & & +1 & \mathbf{-3} & +1 \end{array}\right] \end{array}$$

Suppose that $\theta_3 = -2$, $\theta_4 = -2$, and $\theta_5 = -3$. Thus, the OT learner performs an update, because neither ranking value θ_3 or θ_5 of the two winner-preferring constraints C_3 and C_5 is larger than the ranking value θ_4 of the loser-preferring constraint C_4. Yet, the HG learner does not perform an update, because the weighted sum $\theta_3 - 3\theta_4 + \theta_5 = -2 + 6 - 3$ of the violation differences (44) is strictly positive.

When the OT learner updates its current vector $\boldsymbol{\theta}$ in response to its failure on the current piece of data (44), it demotes the loser-preferring constraint C_4 by 1 and promotes both winner-preferring constraints C_3 and C_5 by a small promotion amount. We have found it convenient to express this promotion amount as c/w in terms of the number w of winner-preferring constraints and a calibration constant c, as in (12a). Assume that the calibration constant c is set equal to $c = 1$. The two winner-preferring constraints C_3 and C_5 are thus promoted by $1/2$ each. Consider next the HG error-driven learner with the original Perceptron reweighing rule (10). This reweighing rule updates each current weight by adding the corresponding constraint difference. Nothing changes in the behavior and the analysis of the learner if we adopt a variant of this reweighing rule whereby the violation differences are rescaled by a (nonnegative) *step size*. For concreteness, set the step size equal to $1/2$. In this case, the reweigh triggered by (44) is as follows: the weight of the constraint C_4 is decreased by adding the corresponding violation difference -2 times the step size $1/2$, namely is decreased by 1; and both weights of constraints C_3 and C_4 are increased by adding the corresponding violation difference 1 times the step size $1/2$, namely are increased by $1/2$. The updates performed by the OT and the HG learners thus turn out to be identical. This conclusion holds in complete generality: since the size of the negative entries in the counterexample (15) is always equal to the number of positive entries, an update by the OT learner with the calibration constant c set equal to $c = 1$ is eqivalent to an update by the HG learner with the original Perceptron reweighing rule (and a properly chosen step size). This conclusion crucially requires the size of the negative entries to be equal to the number of positive entries: no choice of the step size allows the HG update to mimic the OT update in the case of the violation differences (45).

Appendix D. Computing the number of errors on the most favorable training sequence

As in appendix B, let $\mathbf{A}_n$ be the EWC matrix (15a). The final weight vector $\boldsymbol{\theta}$ entertained at convergence by the Perceptron HG learner trained on the counterexample must satisfy the HG consistency condition (8) with each piece of training data. Equivalently, it must satisfy the strict matrix inequality (46a), where $\mathbf{0}$ is a vector of entries all equal to 0. At each update, the reweighing rule (10) adds to each current weight the corresponding violation difference. Equivalently, it adds to the current weight vector the (transposed) row of the matrix $\mathbf{A}_n$ which is the EWC of the piece of data triggering the current update. Since the algorithm starts from the null weight vector, the final weight vector is the sum of the (transposed) rows of the matrix $\mathbf{A}_n$, each multiplied by the corresponding number of updates it has triggered in the run considered. The latter condition can be expressed as the matrix equation (46b), for some integer vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{n-1})$, where α_i represents the number of updates triggered by the EWC corresponding to the ith row of the matrix $\mathbf{A}_n$. By combining (46a) and (46b), I obtain the strict matrix inequality (46c). Finally, since $\boldsymbol{\alpha}$ is an integer vector and $\mathbf{A}_n$ is an integer matrix, the strict inequality (46c) is equivalent to the inequality (46d), where $\mathbf{1}$ is a vector of entries equal to 1.

$$
\begin{aligned}
(46)\quad &\text{a. } \mathbf{A}_n\boldsymbol{\theta} > \mathbf{0} \\
&\text{b. } \boldsymbol{\theta} = \mathbf{A}_n^{\mathrm{T}}\boldsymbol{\alpha} \\
&\text{c. } \mathbf{A}_n\mathbf{A}_n^{\mathrm{T}}\boldsymbol{\alpha} > \mathbf{0} \\
&\text{d. } \mathbf{A}_n\mathbf{A}_n^{\mathrm{T}}\boldsymbol{\alpha} \geq \mathbf{1}
\end{aligned}
$$

The number of updates performed by the HG Perceptron learner thus coincides with the sum $\alpha_1 + \ldots + \alpha_{n-1}$ of the number α_1 of updates triggered by the first EWC plus the number α_2 of updates triggered by the second EWC and so on down to the number α_{n-1} of updates triggered by the last of the $n-1$ EWCs. Furthermore, these nonnegative numbers $\alpha_1, \ldots, \alpha_{n-1}$ need to satisfy the matrix inequality (46d). In the end, the number of updates performed by the learner to converge cannot be smaller than the solution of the optimization problem (47) in the variable $\boldsymbol{\alpha} \in \mathbb{R}^{n-1}$, which therefore provides a lower bound on the *best-case* number of updates performed by the learner.

$$
\begin{aligned}
(47)\quad &\text{minimize:} && \alpha_1 + \ldots + \alpha_{n-1} \\
&\text{subject to:} && \mathbf{A}_n\mathbf{A}_n^{\mathrm{T}}\boldsymbol{\alpha} \geq \mathbf{1} \\
& && \boldsymbol{\alpha} \geq \mathbf{0}
\end{aligned}
$$

As (47) is a linear program, it can be solved with standard optimization software, such as the `Matlab` optimization toolbox. The results for $n = 5, 6, 7, 8, 9, 10$ are plotted in (25).

Appendix E. Simulation details and results

The details of the HG and OT simulations reported in the paper are as follows. The initial weights and initial ranking values are all equal to zero (but for the simulations described in subsection 6.1). The training underlying/winner/loser form triplet are sampled uniformly at random. The re-ranking rule used for the OT simulations is (12) with the calibration constant c set equal to $1/2$. Thus, $w = 3$ winner preferring constraints are promoted by $c/w = 1/6$ each. For both OT and HG stochastic learners, the stochastic values $\epsilon_1, \epsilon_2, \ldots$ are sampled independently and uniformly between $-\Delta$ and $+\Delta$, with $\Delta = 2$. The OT and HG error-driven learners used for the simulations are available as the Python scripts `OTlearner.py` and `HGlearner.py` on the author's webpage, together with the `Excel` files which specify the various counterexamples. The tables which follow provide detailed simulation results.

		I	**II**	**III**	**IV**	**V**	**VI**	**VII**	**VIII**	**IX**	**X**	**mean**	**SD**
$n = 5$	num. of updates	**162**	159	157	156	157	157	162	160	152	165	158.7	3.5
	final weight of C_1	**15**	15	15	15	15	15	15	15	15	15	15.0	0.0
$n = 6$	num. of updates	664	685	668	692	693	686	675	**708**	670	686	682.7	12.8
	final weight of C_1	**31**	31	31	31	31	31	31	31	31	31	31.0	0.0
$n = 7$	num. of updates	2825	2756	2804	2795	2792	2810	2766	**2839**	2806	2791	2798.4	23.5
	final weight of C_1	**63**	63	63	63	63	63	63	63	63	63	63.0	0.0
$n = 8$	num. of updates	11263	11268	11343	11307	11326	11358	11298	11331	11203	**11410**	11310.7	54.5
	final weight of C_1	**127**	127	127	127	127	127	127	127	127	127	127.0	0.0
$n = 9$	num. of updates	45437	45332	45316	**45589**	45524	45230	45403	45451	45411	45535	45422.8	104.3
	final weight of C_1	**255**	255	255	255	255	255	255	255	255	255	255.0	0.0
$n = 10$	num. of updates	182170	181873	182175	182313	182118	181852	182148	**182789**	181710	182331	182147.9	287.9
	final weight of C_1	**511**	511	511	511	511	511	511	511	511	511	511.0	0.0

TABLE 1. Simulation results for ten runs of the deterministic HG learner with the truncated Perceptron reweighing rule on the test case (13).

		I	**II**	**III**	**IV**	**V**	**VI**	**VII**	**VIII**	**IX**	**X**	**mean**	**SD**
$n = 5$	num. of updates	687	668	698	699	**718**	706	689	699	692	690	694.60	12.46
	final weight of C_1	**40**	40	40	40	40	40	40	40	40	40	40.00	0.00
$n = 6$	num. of updates	6153	6113	6151	6143	6140	**6193**	6179	6147	6161	6125	6150.50	22.29
	final weight of C_1	**121**	121	121	121	121	121	121	121	121	121	121.00	0.00
$n = 7$	num. of updates	55171	55230	55152	55005	55179	55082	55221	**55372**	55202	55139	55175.30	91.70
	final weight of C_1	**364**	364	364	364	364	364	364	364	364	364	364.00	0.00
$n = 8$	num. of updates	495445	495486	495452	495714	**496249**	495500	495869	495829	495854	495760	495715.80	241.65
	final weight of C_1	**1093**	1093	1093	1093	1093	1093	1093	1093	1093	1093	1093.00	0.00
$n = 9$	num. of updates	4458363	4457803	4459934	4457700	4459690	4458129	4458814	4460051	4459312	**4460676**	4459047.20	982.80
	final weight of C_1	**3280**	3280	3280	3280	3280	3280	3280	3280	3280	3280	3280.00	0.00

TABLE 2. Simulation results for ten runs of the deterministic HG learner with the truncated Perceptron reweighing rule on a variant of the test case (13) with all entries equal to -2 replaced with -3. The case corresponding to $n = 10$ constraints is omitted because the number of errors becomes too large to run the simulations.

		I	**II**	**III**	**IV**	**V**	**VI**	**VII**	**VIII**	**IX**	**X**	**mean**	**SD**
$n = 5$	num. of updates	4	4	4	4	4	4	4	4	4	4	4.00	0.00
	final weight of C_1	1	1	1	1	1	1	1	1	1	1	1.00	0.00
$n = 6$	num. of updates	5	5	5	5	5	5	5	5	5	5	5.00	0.00
	final weight of C_1	1	1	1	1	1	1	1	1	1	1	1.00	0.00
$n = 7$	num. of updates	6	6	6	6	6	6	6	6	6	6	6.00	0.00
	final weight of C_1	1	1	1	1	1	1	1	1	1	1	1.00	0.00
$n = 8$	num. of updates	7	7	7	7	7	7	7	7	7	7	7.00	0.00
	final weight of C_1	1	1	1	1	1	1	1	1	1	1	1.00	0.00
$n = 9$	num. of updates	8	8	8	8	8	8	8	8	8	8	8.00	0.00
	final weight of C_1	1	1	1	1	1	1	1	1	1	1	1.00	0.00
$n = 10$	num. of updates	9	9	9	9	9	9	9	9	9	9	9.00	0.00
	final weight of C_1	1	1	1	1	1	1	1	1	1	1	1.00	0.00

TABLE 3. Simulation results for ten runs of the deterministic HG learner with the original Perceptron reweighing rule on the test case (13).

			I	**II**	**III**	**IV**	**V**	**VI**	**VII**	**VIII**	**IX**	**X**	**mean**	**SD**
$n = 5$	deterministic	num. of updates	**443**	443	443	443	443	443	443	443	443	443	443.0	0.0
		final weight of C_1	**19**	19	19	19	19	19	19	19	19	19	19.0	0.0
	stochastic	num. of updates	579	526	594	647	**609**	526	443	526	526	526	550.2	54.8
		final weight of C_1	25	23	26	**28**	27	23	19	23	23	23	24.0	2.4
$n = 6$	deterministic	num. of updates	3160	3339	3197	3160	**3391**	3197	3197	3160	3197	3197	3219.5	75.4
		final weight of C_1	51	54	52	51	**55**	52	52	51	52	52	52.2	1.2
	stochastic	num. of updates	4337	4158	3927	4263	4121	**4703**	4084	4300	4263	4084	4224.0	198.3
		final weight of C_1	72	69	65	70	68	**77**	67	71	70	67	69.6	3.2
$n = 7$	deterministic	num. of updates	21961	21871	21460	21243	21871	**21998**	21961	21280	21871	21243	21675.9	309.6
		final weight of C_1	**140**	139	137	135	139	140	140	135	139	135	137.9	2.1
	stochastic	num. of updates	26784	25819	26874	26410	27629	27165	28130	**28220**	27233	26874	27113.8	702.4
		final weight of C_1	171	165	172	169	177	174	180	**181**	174	172	173.5	4.6
$n = 8$	deterministic	num. of updates	133428	135018	136608	135018	131312	137943	134800	133210	133736	**139315**	135038.8	2253.1
		final weight of C_1	339	343	347	343	333	350	342	338	340	**353**	342.8	5.6
	stochastic	num. of updates	168091	177430	176634	172425	178352	**178966**	176275	162614	166936	176994	173471.7	5390.6
		final weight of C_1	429	453	450	439	455	**456**	450	414	426	451	442.3	13.8
$n = 9$	deterministic	num. of updates	821387	820642	**845771**	824031	823813	816636	821695	810395	824340	824648	823335.8	8560.8
		final weight of C_1	843	842	**868**	845	845	838	843	831	846	846	844.7	8.9
	stochastic	num. of updates	989594	979310	1050007	**1077767**	1064913	1037207	1034319	1019037	972787	1059635	1028457.6	35244.2
		final weight of C_1	1018	1007	1080	**1108**	1095	1067	1064	1048	1001	1089	1057.7	36.0
$n = 10$	determ.	num. of updates	4940144	4902760	5000286	**5034378**	5022517	4954640	4966720	4988643	4992242	4954332	4975666.2	37743.6
		final weight of C_1	2065	2049	2090	**2104**	2099	2071	2076	2085	2087	2071	2079.7	15.7
	stochastic	num. of updates	6033904	**6358436**	6071696	6254449	6209458	6259191	6164840	6188381	6102990	6259510	6190285.5	94387.2
		final weight of C_1	2526	**2661**	2541	2618	2599	2620	2580	2590	2554	2619	2590.8	39.4

TABLE 4. Simulation results for ten runs of the deterministic and the stochastic HG learner with the original Perceptron reweighing rule on the counterexample (15).

			I	**II**	**III**	**IV**	**V**	**VI**	**VII**	**VIII**	**IX**	**X**	**mean**	**SD**
$n = 5$	deterministic	num. of updates	570	577	622	**623**	603	590	617	611	605	605	602.3	17.2
		final weight of C_1	26	25	**27**	27	27	26	27	27	27	27	26.6	0.7
	stochastic	num. of updates	655	733	763	708	818	792	643	**932**	834	758	763.6	82.1
		final weight of C_1	30	34	35	32	**37**	37	30	42	39	35	35.1	3.6
$n = 6$	deterministic	num. of updates	4489	4554	**4753**	4682	4550	4566	4707	4686	4556	4558	4610.1	83.5
		final weight of C_1	71	71	**75**	73	72	72	74	74	72	72	72.6	1.3
	stochastic	num. of updates	5278	5714	**5760**	5670	5416	5218	5665	5481	4839	5101	5414.2	287.8
		final weight of C_1	87	93	**94**	92	89	86	93	90	81	84	88.9	4.1
$n = 7$	deterministic	num. of updates	29933	30106	29836	29951	**31304**	29044	29940	29233	29399	29528	29827.4	594.7
		final weight of C_1	179	180	179	179	**186**	175	179	175	176	177	178.5	3.0
	stochastic	num. of updates	34983	37946	**38735**	37450	35611	33588	34651	37592	37343	38620	36651.9	1707.2
		final weight of C_1	215	233	**238**	230	220	208	214	231	230	237	225.6	9.9
$n = 8$	deterministic	num. of updates	181899	186428	180302	**188380**	182778	184518	180485	185289	180126	185619	183582.4	2735.5
		final weight of C_1	429	438	425	**443**	430	434	425	436	425	437	432.2	6.0
	stochastic	num. of updates	228857	209483	220121	221168	210565	**236773**	229481	217572	224811	224176	222300.7	8026.3
		final weight of C_1	550	503	529	531	508	**566**	551	523	540	538	533.9	18.5
$n = 9$	deterministic	num. of updates	1095500	1105544	1091666	1108314	**1110106**	1086011	1101593	1091751	1082304	1098675	1097146.4	8896.3
		final weight of C_1	1039	1048	1036	1050	**1052**	1030	1045	1035	1026	1042	1040.3	8.2
	stochastic	num. of updates	1259561	1363447	1366861	1327487	1329625	1383158	**1387422**	1381051	1316483	1359499	1347459.4	37655.5
		final weight of C_1	1211	1307	1310	1275	1276	1325	**1331**	1324	1264	1304	1292.7	34.9
$n = 10$	determ.	num. of updates	6405051	6374308	6455737	6485087	6462962	6436599	6469502	6456939	**6503850**	6492921	6454295.6	37877.8
		final weight of C_1	2478	2467	2498	2509	2501	2491	2502	2498	**2517**	2512	2497.3	14.5
	stochastic	num. of updates	7860541	7881304	7613735	7670559	7809378	7784654	7832027	7931858	7930757	**7975994**	7829080.7	109578.9
		final weight of C_1	3064	3071	2968	2989	3042	3033	3051	3090	**3090**	3107	3050.5	42.3

TABLE 5. Simulation results for ten runs of the deterministic and the stochastic HG learner with the truncated Perceptron reweighing rule on the counterexample (15).

			I	**II**	**III**	**IV**	**V**	**VI**	**VII**	**VIII**	**IX**	**X**	**mean**	**SD**
$n = 5$	deterministic	num. of updates	**11**	11	11	11	11	11	11	11	11	11	11.00	0.00
		final weight of C_1	**0.17**	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.00
	stochastic	num. of updates	17	14	17	26	32	17	**35**	23	11	17	20.90	7.48
		final weight of C_1	0.33	0.17	0.33	0.50	0.50	0.33	**0.67**	0.50	0.17	0.33	0.38	0.15
$n = 6$	deterministic	num. of updates	**18**	18	18	18	18	18	18	18	18	18	18.00	0.00
		final weight of C_1	**0.17**	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.00
	stochastic	num. of updates	29	36	28	31	31	29	**53**	30	31	36	33.40	7.03
		final weight of C_1	0.33	0.50	0.50	0.50	0.50	0.33	**0.67**	0.50	0.50	0.50	0.48	0.09
$n = 7$	deterministic	num. of updates	**24**	24	24	24	24	24	24	24	24	24	24.00	0.00
		final weight of C_1	**0.17**	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.00
	stochastic	num. of updates	74	70	83	75	**84**	30	83	80	81	51	71.10	16.57
		final weight of C_1	0.67	0.67	**0.83**	0.67	0.83	0.33	0.83	0.83	0.67	0.50	0.68	0.16
$n = 8$	deterministic	num. of updates	**33**	33	31	31	33	33	31	33	31	33	32.20	0.98
		final weight of C_1	**0.17**	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.00
	stochastic	num. of updates	58	77	**137**	78	83	109	135	107	66	60	91.00	27.81
		final weight of C_1	0.50	0.67	0.67	0.67	0.67	**0.83**	0.83	0.67	0.50	0.67	0.67	0.11
$n = 9$	deterministic	num. of updates	**38**	38	38	38	38	38	38	38	38	38	38.00	0.00
		final weight of C_1	**0.17**	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.00
	stochastic	num. of updates	96	146	147	182	121	177	**185**	129	156	144	148.30	26.91
		final weight of C_1	0.67	**0.83**	0.67	0.83	0.67	0.83	0.83	0.67	0.67	0.67	0.73	0.08
$n = 10$	deterministic	num. of updates	47	47	47	47	**49**	47	47	47	47	47	47.20	0.60
		final weight of C_1	**0.17**	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.00
	stochastic	num. of updates	216	171	187	183	221	**231**	169	66	198	222	186.40	45.29
		final weight of C_1	**0.83**	0.67	0.83	0.83	0.83	0.83	0.83	0.50	0.83	0.83	0.78	0.11

TABLE 6. Simulation results for ten runs of the deterministic and the stochastic OT learner on the counterexample (15).

		I	**II**	**III**	**IV**	**V**	**VI**	**VII**	**VIII**	**IX**	**X**	**mean**	**SD**
$n = 5$	num. of updates	**19**	19	19	19	19	19	19	19	19	19	19.0	0.0
	final weight of C_1	**3**	3	3	3	3	3	3	3	3	3	3.0	0.0
$n = 6$	num. of updates	**32**	16	16	16	32	16	16	16	16	16	19.2	6.4
	final weight of C_1	**4**	2	2	2	4	2	2	2	2	2	2.4	0.8
$n = 7$	num. of updates	**37**	37	37	37	37	37	37	37	37	37	37.0	0.0
	final weight of C_1	**3**	3	3	3	3	3	3	3	3	3	3.0	0.0
$n = 8$	num. of updates	**84**	84	84	84	84	84	84	84	84	84	84.0	0.0
	final weight of C_1	**5**	5	5	5	5	5	5	5	5	5	5.0	0.0
$n = 9$	num. of updates	**158**	111	111	111	111	111	111	111	111	111	115.7	14.1
	final weight of C_1	**7**	5	5	5	5	5	5	5	5	5	5.2	0.6
$n = 10$	num. of updates	**206**	206	206	206	206	206	206	206	206	206	206.0	0.0
	final weight of C_1	**7**	7	7	7	7	7	7	7	7	7	7.0	0.0

TABLE 7. Simulation results for ten runs of the deterministic HG learner with the original Perceptron reweighing rule on the test case (23).

		I	**II**	**III**	**IV**	**V**	**VI**	**VII**	**VIII**	**IX**	**X**	**mean**	**SD**
$n = 5$	num. of updates	59	67	59	59	59	67	**79**	67	59	67	64.2	6.2
	final weight of C_1	4	4	4	4	4	4	**5**	4	4	4	4.1	0.3
$n = 6$	num. of updates	98	98	73	**107**	98	100	98	100	98	98	96.8	8.4
	final weight of C_1	3	3	2	3	3	**4**	3	4	3	3	3.1	0.5
$n = 7$	num. of updates	**98**	90	82	82	90	90	82	82	82	82	86.0	5.4
	final weight of C_1	**2**	2	2	2	2	2	2	2	2	2	2.0	0.0
$n = 8$	num. of updates	87	87	100	87	87	**112**	87	87	87	100	92.1	8.4
	final weight of C_1	**2**	2	2	2	2	2	2	2	2	2	2.0	0.0
$n = 9$	num. of updates	90	90	90	90	90	90	90	90	90	90	90.0	0.0
	final weight of C_1	**2**	2	2	2	2	2	2	2	2	2	2.0	0.0
$n = 10$	num. of updates	91	108	91	91	120	**133**	108	91	91	108	103.2	14.1
	final weight of C_1	1	**2**	1	1	2	2	2	1	1	2	1.5	0.5

TABLE 8. Simulation results for ten runs of the deterministic HG learner with the original Perceptron reweighing rule on the test case (24).

		I	**II**	**III**	**IV**	**V**	**VI**	**VII**	**VIII**	**IX**	**X**	**mean**	**SD**
$n = 5$	deterministic	**71**	71	71	71	71	71	71	71	56	71	69.5	4.5
	stochastic	**331**	189	267	234	286	190	183	260	189	209	233.8	48.0
$n = 6$	deterministic	145	145	145	**182**	182	145	145	182	182	182	163.5	18.5
	stochastic	1358	1509	1610	1045	1026	1455	**1727**	891	1438	1344	1340.3	257.2
$n = 7$	deterministic	145	145	145	**1405**	145	1045	1195	992	1292	842	735.1	503.7
	stochastic	7823	7394	7648	8907	8419	8204	7582	8115	6567	**9736**	8039.5	824.7
$n = 8$	deterministic	**9073**	7676	3306	4500	2968	6482	7128	6783	3606	4740	5626.2	1974.6
	stochastic	46614	42298	47394	**65514**	52826	47209	46966	48683	45848	58602	50195.4	6601.7
$n = 9$	deterministic	**32597**	4627	31342	30747	16612	6558	20393	13469	23823	18025	19819.30	9446.25
	stochastic	35572	**317486**	299358	256241	284949	306657	296116	274793	293983	279268	264442.30	78030.15
$n = 10$	deterministic	11410	29211	17704	37574	24945	**64247**	13586	10155	22258	4329	23541.90	16487.24
	stochastic	1559803	1475669	1306498	1481968	**1629844**	1423370	1472857	1463325	1568700	1544966	1492700.00	85415.88

TABLE 9. Number of errors made by the HG learner with the truncated Perceptron reweighing rule in ten runs on the counterexample (15) starting from the initial weights in (30).