ERROR-DRIVEN LEARNING IN HARMONIC GRAMMAR

GIORGIO MAGRI

Abstract — The HG literature has adopted so far the *Perceptron* reweighing rule because of its convergence guarantees. Yet, this rule is not suited to HG, as it fails at ensuring non-negativity of the weights. The first contribution of this paper is a solution to this impasse. I consider a variant of the Perceptron which truncates any update at zero, thus maintaining the weights non-negative in a principled way. And I show that the convergence guarantees for the original Perceptron extend to its truncated variant. Unfortunately, although convergent, HG error-driven learning (with both the original and the truncated Perceptron reweighing rule) is not efficient, contrary to error-driven learning in OT. Indeed, the second contribution of this paper is a counterexample with just ten constraints where the HG learner makes over five million errors before converging while the OT error-driven learner makes less than fifty errors, and yet the HG and OT typologies coincide! The superiority of OT over HG error-driven learning is shown to extend to the stochastic implementation. These results do not contradict the good performance of the Perceptron reported in the Machine Learning and Computational Linguistics literature, as the latter literature has focused on an implementation of the Perceptron which is not error-driven (the *kernel dual Perceptron*), precisely to cope with the inefficiency of the error-driven Perceptron adopted in the HG literature.

Contents

1. Introduction 2
2. Endowing HG with an error-driven learner that keeps the weights non-negative 3
2.1. HG error-driven learning algorithms 3
2.2. The original Perceptron reweighing rule and the problem of non-negative weights 4
2.3. The truncated Perceptron reweighing rule 5
2.4. Summary 7
3. Error-driven learning in HG is much slower than in OT 7
3.1. OT error-driven ranking algorithms 7
3.2. Review of error-bounds for error-driven ranking algorithms 8
3.3. Comparison 9
3.4. Discussion 12
3.5. Summary 13
4. Stochastic error-driven learning is much slower in HG than in OT 13
4.1. Stochastic error-driven learning in HG and OT 13
4.2. Error-bounds for stochastic error-driven learning in HG and OT 14
4.3. Comparison 15
4.4. Summary 17
5. Conclusions 17
Appendix A. HG error-driven learning for general (not necessarily binary) constraints 19
A.1. Description of the learning algorithms 19
A.2. Reformulation in EWC notation 19
A.3. Radius and margin of the training data 20
A.4. Convergence guarantees and error-bounds 22
Appendix B. Simulation details and results 23
B.1. Details concerning the counterexample 23
B.2. Details concerning the OT and HG simulations 24
Appendix C. Technical details 25
C.1. Proof of the convergence theorem 1 for the original Perceptron 25
C.2. Proof of the convergence theorem 2 for the truncated Perceptron 26
C.3. Proof of the convergence theorem 4 for the (original and truncated) stochastic Perceptron 28
C.4. Computing the margin of the counterexample 29
C.5. Computing the best-case number of errors on the counterexample 29
References 30

Part of this paper has been presented at the 21st Manchester Phonology Meeting. I wish to thank Adam Albright, Paul Boersma, and Joe Pater for useful discussion. This research has been supported in part by a grant from the Fyssen Research Foundation.

1. Introduction

An *error-driven* learner is trained on a single piece of data at the time and slightly updates its current hypothesis of the target grammar whenever it makes an error on the current piece of data. Learning continues until no more errors are made and the learner thus settles on a final grammar consistent with the training data. This learning scheme has been endorsed within the language acquisition literature (at least since Wexler and Culicover 1980) because of its good modeling properties. In particular, the learner does not require a lexicon, as each instantaneous update is triggered by the current error on a single piece of data, without paying attention to the implications of that update for the rest of the training data. This learning scheme is thus suited to model also the earliest stages of language acquisition, prior to the development of the native language lexicon. Furthermore, the learning dynamics describes a sequence of grammars which can be matched with child acquisition paths. This learning scheme thus provides a straightforward tool to model the child's acquisition gradualness. This paper looks at error-driven learning within two frameworks for constraint-based phonology, namely *Harmonic Grammar* (HG; Legendre, Miyata, and Smolensky 1998b,a) and *Optimality Theory* (OT; Prince and Smolensky 2004).

Whenever the HG error-driven learner makes an error, the constraints are slightly reweighed. The recent HG computational literature has adopted the *Perceptron* reweighing rule (Jesney and Tessier 2011; Coetzee and Pater 2008, 2011; Coetzee and Kawahara 2013; Boersma and Pater to appear, among many others). According to this rule, a certain amount is added to certain weights and subtracted from others. The main reason for the adoption of this specific reweighing rule is that it comes with guarantees that the number of errors is always finite, so that the learner provably *converges* to a grammar consistent with the data and learning ceases (Block 1962; Novikoff 1962; Rosenblatt 1958, 1962; Minsky and Papert 1969; Cesa-Bianchi and Lugosi 2006, chapters 11, 12; Mohri et al. 2012, ch. 7). Yet, HG crucially requires the weights to be non-negative, in order to avoid undesired typological predictions. Unfortunately, there is just no way to guarantee non-negativity of the final weights in the case of the Perceptron, which therefore does not qualify as a proper HG error-driven learner, despite current practice. Section 2 offers a solution to this problem. I consider a *truncated* version of the Perceptron reweighing rule, which is identical to the original Perceptron rule, but for the fact that a weight is not updated when an update would otherwise make it negative, thus ensuring non-negativity of the final weights. Despite the fact that a run of the original and the truncated Perceptron can differ substantially, I note that a run of the truncated Perceptron can always be mimicked through a run of the original Perceptron on a properly extended set of data. I thus conclude that convergence guarantees extend from the original to the truncated Perceptron.

Section 3 then compares the HG and OT implementations of error-driven learning from the perspective of *error-bounds*, namely the worst-case number of errors made by the learner before convergence (as a function of the number of constraints). For the case of OT, the available error-bound grows slowly (namely, quadratically) with the number of constraints, for a variety of different implementations (Tesar and Smolensky 1998; Magri 2012b,a, 2013a). The error-bound for the HG learner with the (original and truncated) Perceptron reweighing rule instead grows very fast (namely, exponentially). So fast that I can construct a case with just ten constraints where the HG error-driven learner makes over five million errors — contrary to less than fifty errors made by the OT error-driven learner. Importantly, the constraint set considered in this counterexample has the property that the corresponding typologies according to HG and OT exactly coincide. The inefficiency of the HG implementation thus cannot be due to the fact that the algorithm is exploring a larger typology. In other words, even if the typologies predicted for a given constraint set 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 still outperforms the HG one from the perspective of error-driven learning. Finally, I explain that this conclusion concerning the superiority of the OT over the HG implementation of error-driven learning is not at odds with the good performance of the Perceptron reported in the Machine Learning (e.g., Mohri et al. 2012, ch. 7) and Computational Linguistics literature (e.g., Collins 2002). In fact, the latter literature has focused on an implementation of the Perceptron (the *kernel dual Perceptron*) which has access to the entire batch of data at once and is therefore not error-driven, precisely in order to cope with the inefficiency of the classical error-driven Perceptron adopted in the HG literature.

Section 4 compares HG and OT error-driven learning further, looking at the stochastic implementation. A stochastic error-driven learner randomly perturbs the current grammar before testing it on the current piece of data (Boersma 1997, 1998; Boersma and Hayes 2001; Coetzee and Pater 2008, 2011; Coetzee and Kawahara 2013; Boersma and Pater to appear). Convergence and error-bounds extend straightforwardly to the stochastic implementation, both for the case of HG (Boersma and Pater to appear) and OT (Magri 2013a). Thus again, the error bound for the OT stochastic error-driven learner grows slowly (namely, quadratically) with the number of constraints while the error-bound for the HG stochastic learner grows very fast (namely, exponentially). So fast that with just ten constraints, the HG stochastic learner makes over one million additional errors on top of the already astronomical number of errors made by its deterministic counterpart, while the OT stochastic learner makes around 150 additional errors on top of the already very small number of errors made by its deterministic counterpart. I thus conclude that OT outperforms HG also from the perspective of the stochastic implementation of error-driven learning.

A crucial component of the language acquisition task is that of finding a grammar consistent with the entire batch of data. Error-driven learning needs to achieve this goal with extremely impoverished computational resources. In fact, contrary to competing more powerful learning algorithms, the error-driven learner does not have access to the

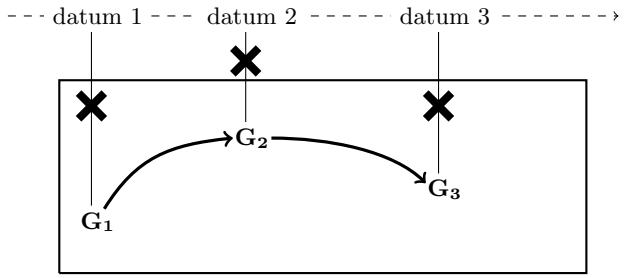
entire batch of data at once. The learning dynamics is driven by instantaneous updates each triggered by a single piece of data at the time, without ever being able to compute the implications of the current update for the entire batch of training data. The modest computational resources used by error-driven learning make it suitable to model also early acquisition stages, while the computational resources required by alternative more powerful algorithmic schemes would be unrealistic at this early stage. For instance, since the error-driven learner does not keep track of previously seen data, it is perfectly suited to model the early stages of the acquisition of phonotactics, prior to the acquisition of the native language lexicon (Hayes 2004). On the other hand, because of its modest computational resources, error-driven learning can succeed at the learning task only if implemented with extreme care. Within constraint-based phonology, this means that it can succeed only if implemented within a carefully crafted model of constraint interaction, simple enough to boost the modest resources of this learning scheme. Section 5 concludes the paper, suggesting that the OT model of constraint interaction is superior to the HG model from the perspective of boosting the efficiency of the error-driven model of child language acquisition.

These final considerations show that the computational and simulation results presented in the paper have broad implications for the architecture of constraint-based phonology. In order to make these results and their implications accessible also to a non-computational readership, the paper is written in "three layers": the bulk of the paper is kept completely informal; appendices A-B provide more formal details, but do not require any technical background; finally, appendix C contains all technical details and presupposes familiarity with basic Linear Algebra.

2. Endowing HG with an error-driven learner that keeps the weights non-negative

Assume that the learner is provided with the space of all possible grammars G_1, G_2, etcetera. Data come in a stream, one piece of data at the time. The learner maintains a current grammar, which represents its current hypothesis on the target adult grammar. Whenever the current grammar (say G_1) fails to account for the current piece of data (say, datum 1), the learner slightly updates its current grammar to a slightly different one that sits nearby in the space of grammars (say, G_2). This process is repeated until the learner stops making errors and converges to a final grammar consistent with the stream of training data, so that learning ceases.

(1)

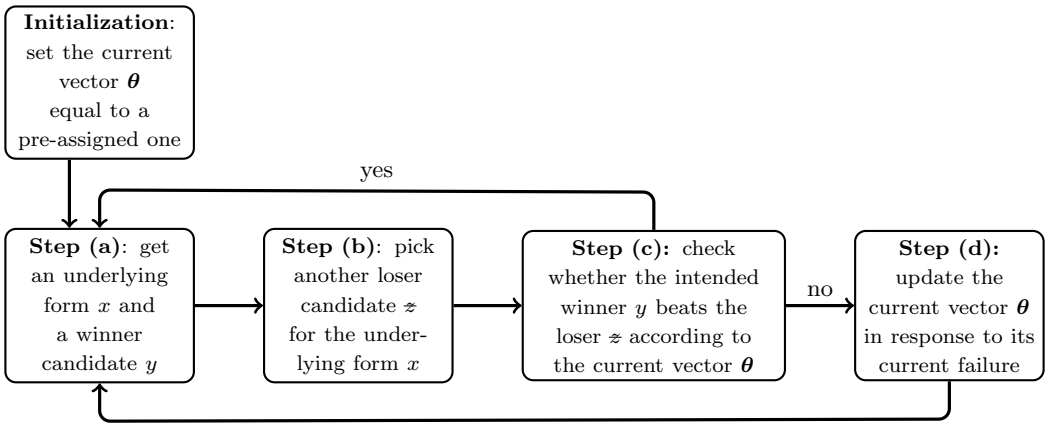


This learning scheme is called *error-driven* because the learning dynamics is driven by the errors performed on the incoming stream of data. This scheme has been thoroughly investigated in the Machine Learning literature (under the heading of *online learning*; for a review, see Kivinen 2003; Cesa-Bianchi and Lugosi 2006, chapters 11, 12; and Mohri et al. 2012, ch. 7). Within the linguistic literature, error-driven learning dates back to at least Wexler and Culicover (1980). This section discusses the proper implementation of error-driven learning within HG.

2.1. HG error-driven learning algorithms

Within HG, the typology of grammars is parameterized by an assignment of *weights* $\theta_1, \ldots, \theta_n$ to a given set of n phonological constraints $C_1, \ldots, C_n$. These weights are collected together into an n-tuple $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$, called a *weight vector*. The informal algorithmic scheme (1) can then be made explicit as in (2).

(2)



The algorithm maintains a current HG grammar, represented through a current vector $\boldsymbol{\theta}$ of weights. These current weights are initialized by setting them equal to certain initial values. For concreteness, I assume throughout this paper that the initial weights are all null. These initial weights are then updated by looping though the four steps (2a)-(2d), described below.

At step (2a), the algorithm receives a piece of data sampled from the target HG grammar the algorithm is being trained on. This piece of data consists of an underlying form x together with the candidate y it is mapped to according to that target HG grammar the algorithm is being trained on. This means in turn that the target HG grammar is such that y wins the competition over any other candidate for that underlying form x. In order to check the performance of the (HG grammar corresponding to the) current weights on this current piece of data, at step (2b) the algorithm picks a loser candidate $\not{y}$ for comparison with the intended winner y. As a mnemonic, I strike out a candidate when it is meant to be a loser. I make no assumptions on how the current loser $\not{y}$ is chosen at step (2b).[1]

At step (2c), the algorithm checks whether indeed the current weights manage to make the intended winner y beat the intended loser $\not{y}$. To keep things simple, I assume throughout this paper that all constraints are binary: they assign either no violations or just one violation; as shown in appendix A, this restriction is easily relaxed, at the only expenses of a slightly more cumbersome notation. With this simplification, the condition that the winner y beats the loser $\not{y}$ according to the current weights $\boldsymbol{\theta} = (\theta_1, \dots, \theta_n)$ boils down to condition (3). Here, W (and L) is the set of *winner-preferring* (*loser-preferring*, respectively) constraints, namely those constraints that assign less (more, respectively) violations to the intended winner y than to the intended loser $\not{y}$.

$$(3) \quad \sum_{h \in W} \theta_h > \sum_{k \in L} \theta_h$$

This condition (3) says that the constraints which prefer the winner y "outperform" the constraints which instead prefer the loser $\not{y}$, in the sense that the sum of the weights of the former constraints is strictly larger than the sum of the weights of the latter constraints.

If condition (3) is satisfied at step (2c), then the current weights manage to make the intended winner y beat the loser $\not{y}$. Hence, the algorithm has nothing to learn from this comparison, loops back to step (2a) and waits for more data. Otherwise, the algorithm needs to take action, by slightly revising the current weights at step (2d). Failure of condition (3) suggests that the weights corresponding to the winner-preferring (loser-preferring) constraints are too small (too large, respectively). One reasonable update strategy is thus (4): the weights corresponding to the winner-preferring (loser-preferring) constraints are increased (decreased, respectively) by a small amount, say 1; for an illustration, see (9) below.

(4) a. Increase the current weight of each winner-preferring constraint by 1;
b. decrease the current weight of each loser-preferring constraint by 1.

The algorithmic scheme (1) with the update condition (3) at step (1c) and a reweighing rule such as (4) at step (1d) is called an HG *error-driven learning algorithm*.

2.2. The original Perceptron reweighing rule and the problem of non-negative weights

Boersma and Pater (to appear) note that the HG error-driven learner with the specific update rule (4) is known in the Machine Learning literature as the *Perceptron* algorithm for linear classification (for the case of binary constraints). The algorithm comes with convergence guarantees summarized in theorem 1 (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). Appendices A.1 and A.2 describe the Perceptron algorithm with arbitrary (namely, not necessarily binary) constraints; appendix A.3 formally defines the notion of *margin* which appears at the denominator of (5); appendix A.4 provides a general statement of the theorem for arbitrary constraints; finally, appendix C.1 recalls the proof of the theorem for completeness.

Theorem 1. *The HG error-driven learner (2) with the Perceptron reweighing rule (4) converges: whenever trained on data consistent with some HG grammar, it can only perform a finite number of errors, before settling on final weights which are guaranteed to be consistent with the target grammar, so that no further updates are triggered and learning ceases. Furthermore, the number of errors made before converging can be bound as follows*

$$(5) \quad \textit{number of errors} \leq \frac{n^2}{\big(\textit{margin of the training data}\big)^2}$$

in terms of the number n of constraints and a quantity that captures a crucial property of the training data, namely their margin.[2] ■

[1] A reasonable choice is to set the current loser $\not{y}$ equal to the candidate which is predicted to win according to the current weights $\boldsymbol{\theta}$. With this definition of step (2b), the following step (2c) can be reformulated as follows: "check whether the intended winner y coincides with the predicted winner $\not{y}$".

[2] The general error-bound for the Perceptron algorithm has the squared radius of the data at the numerator, not the squared number of constraints, as in (5). Yet, since I am restricting myself to binary constraints, the squared radius is always bounded by the squared number of constraints; see appendix A.3. Indeed, it is the margin (not the radius) which will be held responsible for the exponential

Theorem 1 crucially assumes that the training data are all consistent with a certain HG grammar. It turns out that consistent training datasets can differ because of their "degree of consistency". The margin which appears in the denominator of the error-bound (5) is indeed a measure of the degree of consistency of the training data, as explained in appendix A.3. Training data with a higher degree (lower degree) of consistency have a larger (smaller, respectively) margin, yielding a smaller and thus better (larger and thus worse, respectively) error-bound (5). Section 3 will discusses the quality of the error-bound (5) and its dependence on the margin of the data.

Unfortunately, the use of the Perceptron as an HG error-driven learner suffers from the following well-known problem. In order for HG to avoid undesired typological predictions, constraint weights need to be enforced to satisfy the non-negativity condition (6).

(6) $\theta_1, \ldots, \theta_n \geq 0$

Here is an elementary counterexample which illustrates the importance of this non-negativity condition. Consider a constraint set which consists of a markedness constraint C_1 against voiced obstruents and a faithfulness constraint C_2 that enforces identity for voicing. The set of underlying forms contains the voiceless stop /ta/. The generating function pairs it up with the two candidates [ta] and [da]. If the two constraints C_1 and C_2 are allowed to take on negative weights (say $\theta_1 = -3$ and $\theta_2 = -1$), then the corresponding HG grammar maps the voiceless stop to a voiced one. This contradicts a crucial tenet of constraint-based phonology: unfaithful mappings should always yield a gain in markedness.

Despite the fact that the non-negativity condition (6) is crucial from the perspective of HG's typological predictions, the Perceptron update rule (4) used in the current HG literature does not in any way guarantee that the current and final weights entertained by the algorithm satisfy this non-negativity condition (6). Even if the current weights are initialized to large initial values, there is no guarantee that they will never drop below zero, as the number of updates — and thus in particular the number of demotions (4b) — crucially depends on the size of the initial weights. Furthermore, certain modeling applications have been argued to require certain constraints to start with null initial weights, namely to start right on the edge of the forbidden zone (for instance, Jesney and Tessier 2011 argue that input-output faithfulness constraints need to start with null initial weights, in order to prevent gang up effects that would lead to phonotactically unrestrictive final rankings). In conclusion, the Perceptron reweighing rule (4) does not yield a proper HG error-driven learner. The rest of this section develops a solution to this problem.[3]

2.3. **The truncated Perceptron reweighing rule**

I propose to switch from the original Perceptron update rule (4) to the *truncated* Perceptron update rule (7). The two update rules coincide as long as the current weights stay non-negative. But when the original update rule (4) would demote a certain weight below zero, the truncated rule (7) leaves that weight unchanged;[4] for an illustration, see below (10).

(7) a. Increase the current weight of each winner-preferring constraint by 1;
b. decrease the current weight of each loser-preferring constraint by 1...
... unless that would make that weight negative, in which case do not modify that weight.

Theorem 1 guarantees that the original Perceptron (4) can only make a finite number of errors and furthermore provides an error-bound in terms of certain geometric properties of the data. What about the truncated Perceptron (7)? Suppose that the current weights are all initialized to zero. The Perceptron update rule will then perform lots of demotions below zero that the truncated Perceptron is forbidden to mimic. As a result, the learning dynamics of the original Perceptron will turn out to be quite different from the learning dynamics of the truncated Perceptron. Is there any way to extend the theoretical guarantees that hold for the original Perceptron to its truncated variant?

To gain some insight into the problem, let's consider a concrete example. Suppose that the constraint set consists of the three constraints C_1, C_2, and C_3 in (8a). Constraints C_1 and C_2 are faithfulness constraints for voicing,

---

growth of the Perceptron error-bound. The formulation of the bound in (5) thus allows me to focus on the crucial characters (namely, the margin and the number of constraints), leaving the details for appendix A.4, which recalls the general error-bound for arbitrary (not necessarily binary) constraints.

[3] A different solution to this problem is to use the *Winnow* instead of the Perceptron algorithm (Littlestone 1988). In fact, Winnow adopts a multiplicative update rule (rather than the Perceptron's additive update rule) and therefore effectively keeps the weights non-negative. Yet, convergence guarantees for Winnow only hold when the amount of reweighing (also called the *plasticity* or the *step size* of the reweighing rule) has been properly chosen in a way that crucially depends on certain geometric properties of the training data, namely their *margin*. Since of course the margin is not known beforehand, then the algorithm needs to be supplemented with a procedure to estimate the margin online, making the overall implementation more complex. Despite this difficulty, it might be worth exploring the use of the Winnow's reweighing rule for HG error-driven learning. In fact, Boersma and Pater (to appear) report simulation results with a reweighing rule which is very similar to Winnow's one (it only differs because the current weights are not normalized, contrary to what prescribed by Winnow). Although the variant tested in Boersma and Pater's simulations has no guarantees of convergence (normalization of the weights plays a crucial role in the Winnow's convergence proof), they report that the number of errors is significantly smaller than in the case of the Perceptron. Indeed, Winnow and the Perceptron have been compared extensively in the Machine Learning literature (Kivinen, Warmuth, and Auer 1997), with the two update rules outperforming each other on different types of data sets.

[4] A slight variant of (7b) is as follows: when decreasing a weight by 1 would make that weight negative, instead of leaving that weight unchanged, set that weight equal to the smallest licit value, namely to zero. The analysis presented below trivially extends to this variant as well.

with C_1 protecting obstruents in onset positions and C_2 protecting obstruents in both onset and coda position. Constraint C_3 is a markedness constraint which militates against obstruent voicing. To keep things simple, assume that the generating function is only allowed to change obstruent voicing. Thus, the underlying form /da/ comes with only two candidates [da] and [ta], as stated in (8b).

(8) a. $\left\{\begin{array}{l} C_1 = \textsc{Ident}[\textsc{voice}]/\textsc{Onset} \\ C_2 = \textsc{Ident}[\textsc{voice}] \\ C_3 = *[+\textsc{voice}] \end{array}\right\}$ b. $Gen(/\text{da}/) = \{[\text{da}], [\text{ta}]\}$

Suppose that at a certain iteration of the HG learner, the current weight of constraints C_1 is null, thus barely satisfying the non-negativity condition (6). Suppose instead that the current weights of constraints C_2 and C_3 are positive, say equal to 7 and 3 respectively, as in the weight vector on the left hand side of (9). Suppose furthermore that the target grammar the learner is trained on does not allow for voiced obstruents, not even in onset position. At the current iteration of step (2a), the learner is thus fed the underlying form /da/ together with the corresponding intended winner [ta], as indicated by the label on top of the arrow in (9). As the only other candidate for this underlying form is the faithful form [da], the algorithm picks the latter as the intended loser at step (2b). The markedness constraint C_3 is winner-preferring in this case and its current weight $\theta_3 = 3$ is not larger than the sum $\theta_1 + \theta_2 = 7$ of the weights of the two loser-preferring faithfulness constraints C_1 and C_3. Condition (3) therefore fails in this case and the learner needs to update its current weights at step (2d).

(9) $\begin{array}{l} C_1 \\ C_2 \\ C_3 \end{array}\begin{bmatrix} 0 \\ 7 \\ 3 \end{bmatrix} \xrightarrow{(/\text{da}/,\ [\text{ta}])} \begin{bmatrix} -1 \\ 6 \\ 4 \end{bmatrix}$

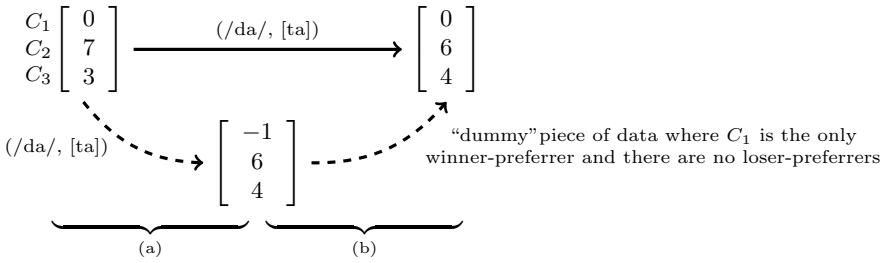
The original Perceptron update rule (4) prescribes that the the weights of the two loser-preferring constraints C_1 and C_2 each be decreased by 1 while the weight of the winner-preferring constraint C_3 be increased by 1, obtaining the updated weight vector on the right hand side of (9). As a result of this update, the weight of the loser-preferring constraint C_1 has dropped down to the negative value $\theta_1 = -1$, violating the non-negativity condition (6). If this were the final update performed by the learner, then it would have effectively learned completely useless weights.

The update according to the truncated Perceptron update rule (7) in the scenario just considered would instead go as in (10). The weight of the winner-preferring constraint C_3 is increased by 1 and the weight of the loser-preferring constraint C_2 is decreased by 1, just as in the case of the original Perceptron. The crucial difference is that the weight of constraint C_1 is left unchanged in order to prevent that weight from turning negative, despite the fact that constraint C_1 is loser-preferring.

(10) $\begin{array}{l} C_1 \\ C_2 \\ C_3 \end{array}\begin{bmatrix} 0 \\ 7 \\ 3 \end{bmatrix} \xrightarrow{(/\text{da}/,\ [\text{ta}])} \begin{bmatrix} 0 \\ 6 \\ 4 \end{bmatrix}$

Crucially, the update (10) by the truncated Perceptron can be perfectly mimicked with two updates by the original Perceptron, as in (11). At the first update (11a), we run the original Perceptron on the current piece of data just as in (9). Thus in particular, the weight of the loser-preferring constraint C_1 is demoted to -1, as in (11a).

(11)



Immediately afterwards, we feed the original Perceptron with a "dummy" piece of data whereby constraint C_1 is the only winner-preferring constraint and there are no loser-preferring constraints. The update condition (3) fails: the right hand side is null (because there are no loser-preferring constraints) and the left-hand side is negative (because C_1 is the only winner-preferring constraint, and its current weight $\theta_1 = -1$ is negative). The original Perceptron thus updates these current intermediate weights as in (11b): the weight of C_1 is increased by 1 back to zero, and no other weights are modified.

The reasoning just illustrated on a concrete example extends rather straightforwardly to the general case. An update according to the truncated Perceptron reweighing rule (7) can thus always be mimicked through two updates according to the original Perceptron reweighing rule (4), namely the update triggered by the actual piece of data followed by another update triggered by a *dummy* piece of data whose only purpose is to undo the illicit demotions that yielded negative weights. Of course, these dummy pieces of data have no phonological meaning. They are just artificial data used in the analysis (not in the simulations!) of the truncated Perceptron reweighing rule.

These dummy pieces of data have the property that they have winner-preferring constraints but no loser-preferring constraints. They thus turn out to be always consistent with the actual data: if the actual training data is consistent with some HG grammar, adding these dummy data does not disrupt consistency (see appendix C.2 for details).

Let me take stock. The convergence theorem 1 for the original Perceptron reweighing rule ensures convergence whenever the training data are consistent. A run of the truncated Perceptron can be mimicked by a run of the original Perceptron on the training data extended with dummy data. Furthermore, this extended set of data is consistent. The convergence theorem 1 for the original Perceptron thus yields the analogous convergence theorem 2 for the truncated Perceptron. Appendix A.1 describes the truncated Perceptron algorithm with arbitrary (namely, not necessarily binary) constraints; appendix A.4 provides a general statement of theorem 2; and appendix C.2 formalizes the reasoning sketched above into an actual proof.

Theorem 2. *The HG error-driven learner (2) with the truncated Perceptron reweighing rule (7) converges: whenever trained on data consistent with some HG grammar, it can only perform a finite number of errors, before converging to* non-negative *final weights which are guaranteed to be consistent with the target grammar, so that no further updates are triggered and learning ceases. Furthermore, the number of errors made before converging can be bound as follows*

$$\text{(12)}\quad \textit{number of errors} \leq \frac{n^2}{\big(\textit{margin of the training plus dummy data}\big)^2}$$

in terms of the number n of constraints and a quantity that captures the crucial geometric property of the training plus dummy data, namely their conjoined margin. ■

The error-bound (12) for the truncated Perceptron depends on the margin of the training plus dummy data while the error-bound (5) for the original Perceptron depends on the margin of the training data only. As it is clear from the geometric definition of the margin provided in Appendix A.3, the margin of the training plus dummy data is in general smaller than or at most equal to the margin of the training data only. This means in turn that the error-bound (12) for the truncated Perceptron is worse than (namely, at least as large as) the error-bound (5) for the original Perceptron. This is consistent with the fact that the truncated Perceptron performs more errors than the original Perceptron in the simulations reported below in section 3.

2.4. **Summary**

This section has addressed the issue of the proper implementation of the error-driven learning scheme (1) within the HG version of constraint-based phonology. Different implementations of HG error-driven learning differ for their reweighing rule. The standard choice in the current HG computational literature is the Perceptron reweighing rule. Yet, that reweighing rule is not suited to HG, as it does not guarantee non-negativity of the weights, which is a crucial ingredient of the definition of HG grammars. I have thus considered a variant of the original Perceptron reweighing rule, whereby the updates are "truncated" at zero, thus enforcing non-negativity of the current and final weights. I have then argued that convergence guarantees of the original Perceptron extend to the truncated variant, as a run of the latter can be mimicked by a run of the former on that same training data extended with certain dummy data used to "undo" the truncated updates. Now that HG has been endowed with a proper error-driven learner, we can ask the question of its computational quality in comparison with error-driven learning within other frameworks for constraint-based phonology, such as OT. This is the goal of the rest of this paper.

3. Error-driven learning in HG is much slower than in OT

In this section, I briefly recall the OT implementation of the error-driven learning scheme (1) and compare it with the HG implementation developed in the preceding section from the perspective of converge and error-bounds.

3.1. **OT error-driven ranking algorithms**

Within OT, the typology of grammars is parameterized by rankings over a given set of n phonological constraints $C_1, \ldots, C_n$. As noted in Boersma (1997, 1998), rankings can be given a numerical representation, as follows. Each constraint C_k is assigned a numerical *ranking value* θ_k. These ranking values are collected into an n-tuple $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$, called a *ranking vector*. A constraint ranking $\gg$ is consistent with a ranking vector $\boldsymbol{\theta}$ provided it satisfies the ordering induced by the relative size of the ranking values, in the sense that a constraint C_h is ranked above a constraint C_k according to $\gg$ whenever the ranking value θ_h of the former is larger than or equal to the ranking value θ_k of the latter. Once rankings are given such a numerical representation, the typology of OT grammars can be parameterized through ranking vectors. The informal error-driven learning scheme (1) can thus be formalized within OT with the same algorithmic scheme (2) used for the HG implementation. Obviously, the current numerical vector $\boldsymbol{\theta}$ entertained by the algorithm is interpreted differently, namely not as the weight vector corresponding to an HG grammar but as the ranking vector corresponding to (a ranking corresponding to) an OT grammar. As a consequence of this different interpretation, the update condition checked at step (2c) and the update rule used at step (2d) are different in the OT implementation, as detailed below.

At step (2c), the algorithm checks whether its current hypothesis manages to make the intended winner y beat the intended loser $\not{z}$. In terms of rankings, this condition requires the top ranked among the winner-preferring

constraints to be ranked above the top ranked among the loser-preferring constraints. Translated in terms of ranking values, this condition requires the largest ranking value among winner-preferring constraints to be larger than the largest ranking value among loser-preferring constraints, as stated in (13), where again W (and L) stands for the set of winner-preferring (loser-preferring, respectively) constraints.

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

This condition (13) says that the constraints which prefer the winner y "outperform" the constraints which instead prefer the loser $\tilde{x}$, in the sense that the largest ranking value among the former constraints is strictly larger than the largest ranking value among the latter constraints (see also Boersma 2009).

Failure of condition (13) suggests that the ranking values of the loser-preferring (winner-preferring) constraints are too large (too small, respectively). Following Tesar and Smolensky (1998), Boersma (1998, p. 323-327), and Magri (2012b), we thus entertain the update strategy in (14): the winner-preferring constraints are promoted by a small *promotion amount* while the loser-preferring constraints are demoted by a small *demotion amount*. What really matters is not the actual values of the promotion and demotion amounts, but rather their ratio. Thus, I set the demotion amount equal to 1, and let the promotion amount take on an arbitrary non-negative value $p \geq 0$. Crucially, the demotion component (14b) of the re-ranking rule does not demote all loser-preferring constraints, but only those that really need to be demoted, namely those loser-preferring constraints which are not currently ranked underneath a winner-preferring constraint and are therefore called *undominated*.

(14) a. Increase the current ranking value of each winner-preferring constraint by $p \geq 0$;
b. decrease the current ranking value of each *undominated* loser-preferring constraint by 1.

The algorithm (2) with the update condition (13) at step (2c) and the re-ranking rule (14) at step (2d) is called an OT *error-driven ranking algorithm*.

Let me take stock. Error-driven learning in OT and HG boils down to the same algorithmic scheme (2). The two implementations only differ for the update condition used at step (2c) and the update rule used at step (2d). Under the assumption that the constraints are binary, the update condition used in the HG implementation boils down to (3), which only differs from the update condition (13) used in the OT implementation because the former uses the sum operator ($\sum$) while the latter uses the maximum operator (max). Furthermore, under the assumption that the constraints are binary, the (original or truncated) Perceptron update rule used in the HG implementation boils down to (4)/(7), which again differs only minimally from the update rule (14) used in the OT implementation. The first difference between the two update rules is that all loser-preferring constraints are treated on a par in the HG implementation, while in OT we distinguish between dominated and undominated loser-preferrers. The second difference is that the promotion amount is set once and for all equal to the demotion amount in the HG implementation, while it needs to be carefully calibrated in the case of OT (see below theorem 3). If the constraints are not binary, then this strong parallelism between OT and HG error-driven learning is broken, as the HG update condition (3) and the HG update rule (4)/(7) take on the more complex shape in (33) and (34) in appendix A, which involve the actual numbers of constraint violations.

3.2. **Review of error-bounds for error-driven ranking algorithms**

In section 2, we have looked at convergence guarantees and error-bounds for error-driven learning in HG. This subsection reviews what is currently known concerning error-driven learning in OT. At each iteration, let w be the number of winner-preferring constraints, which will be promoted by the promotion component (14a) of the re-ranking rule. Let ℓ be the number of loser-preferring constraints undominated according to the current ranking values, which will be demoted by the demotion component (14b) of the re-ranking rule. Let the *calibration threshold* be the ratio ℓ/w between these two numbers. The following theorem 3 says that the OT error-driven learner converges efficiently if and only if the promotion amount p used in the promotion component (14a) of the re-ranking rule is *calibrated*, namely it is strictly smaller than this calibration threshold. In this case, the error-bound (15) grows quadratically in the number n of constraints. Theorem 3 thus provides an exhaustive theory of convergence for OT error-driven learning.

Theorem 3. (I) *If the promotion amount is strictly smaller than the calibration threshold (i.e., $p < \ell/w$), then the OT error-driven learner converges: whenever trained on data consistent with some OT grammar, it can only perform a finite number of errors, before settling on a final ranking vector. Any constraint ranking which is consistent with these final ranking values (namely, which satisfies the ranking conditions induced by their relative size) is consistent with the target grammar. Furthermore, the algorithm is efficient, as the number of errors made before converging can be bound as follows:*

(15) $$\textit{number of errors} \leq \frac{n^2}{\textit{calibration of the promotion amount}}$$

in terms of the number n of constraints and a quantity that depends on the promotion amount, called its calibration. **(II)** *If the promotion amount coincides with the calibration constant (i.e., $p = \ell/w$), then the OT error-driven learner again converges, namely can only make a finite number of errors when trained on data consistent with some OT grammar. Yet, efficiency is lost: it is possible to construct cases where the number of errors grows exponentially*

with the number of constraints, so that the algorithm is unfeasible for realistic constraint sets. **(III)** *If the promotion amount is strictly larger than the calibration constant (i.e., $p > \ell/w$), then converge is lost as well: it is possible to construct cases where the OT error-driven learner makes an infinite number of errors.* ■

Claim **(I)** for $p = 0$ is due to Tesar and Smolensky (1998) (see also Boersma 1997, pp. 323–327 and Boersma 2009); its extension to the case $p < \ell/w$ is due to Magri (2012b,a), together with claim **(II)**; finally, claim **(III)** is due to Pater (2008). See Appendix B.2 for a more precise formulation of the error-bound (15). The latter error bound is tight for a null promotion amount $p = 0$; Magri (2013a) provides improved error-bounds for the case $p \neq 0$ under additional assumptions on the training data. Theorem 3 analyzes error-driven learning under the idealized assumption that the data fed to the algorithm are all consistent with at least a grammar in the typology. A more realistic setting needs of course to allow for the possibility that a small portion of the data fed to the algorithm could have been corrupted by transmission noise or speech errors. Magri (2013a) shows that in this case the number of errors made by the OT error-driven learner with a calibrated re-ranking rule can be bound as follows:

$$\text{(16)} \qquad \text{number of errors} \;\leq \underbrace{\frac{n^2}{\text{calibration}}}_{\text{(a)}} + \underbrace{\frac{n}{\text{calibration}} \times \text{number of corrupted data}}_{\text{(b)}}$$

in terms of the number n of constraints, the number of corrupted data, and the calibration of the promotion amount used in the re-ranking rule. The term (16a) coincides with the bound (15) on the number of errors in the consistent setting and the term (16b) thus expresses the additional number of errors due to the corrupted data. Crucially, this additional term grows only linearly in the number n of constraints, ensuring that the learner is robust to noise.

3.3. **Comparison**

Error-driven learning is an algorithmic scheme with quite limited resources: the learner has access only to one piece of data at the time and needs to make instantaneous decisions without being able to evaluate the implications of those decisions for the rest of the training data. For this reason, the learner is allowed to make a certain number of errors before eventually succeeding. The number of errors quantifies the intrinsic difficulty of error-driven learning. Converge guarantees ensure that the number of errors made by the learner cannot be infinite. Yet, there is little practical difference between the number of errors being infinite and it being finite but astronomically large: in either case, the algorithm is useless, both from a computational and a modeling perspective. The fundamental questions addressed by the theory of error-driven learning in constraint-based phonology can thus be informally stated as follows: how large is the number of errors made by the learner? Does the number of errors depend on the choice between the HG and the OT implementation of error-driven learning? In that case, which of the two implementations makes the smallest number of errors and thus copes best with the intrinsic difficulty of error-driven learning?

Here is a way to state these questions explicitly. 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 the complexity of the learning task is plausibly the number of constraints that the learner needs to rerank or reweigh. In principle, two scenarios might arise. According to one scenario, the number of errors grows as a function of the number of constraints but only slowly (namely, polynomially), so that the number of errors remains feasible also for realistically large constraint sets. According to the opposite scenario, the number of errors grows instead very fast (namely, exponentially) with the number of constraints, so that the number of errors is already astronomically large for realistic constraint sets. The fundamental questions of the theory of error-driven learning in constraint-based phonology can thus be formalized as follows: how fast does the number of errors grow with the number of constraints in the case of the OT and the HG implementation of error-driven learning? which of the two implementations yields the slowest growth of the number of errors and thus copes best with the intrinsic difficulty of error-driven learning?

Let me start with the case of the OT error-driven learner. Theorem 3 provides an explicit bound (15), repeated in (17), on the number of errors that the algorithm can make, even in the worst-case scenario of the most adversarial training sequence. This bound is the product of two factors. The first factor (17a) is the square of the number n of constraints. The second factor (17b) involves a constant which captures a crucial property of the promotion amount, namely its calibration.

$$\text{(17)} \quad \text{number of errors made by the OT learner} \leq \underbrace{n^2}_{\text{(a)}} \times \underbrace{\frac{1}{\text{calibration of the promotion amount}}}_{\text{(b)}}$$

The calibration constant which appears in the second factor (17b) of the OT error-bound does not depend on the data and thus in particular does not depend on the number n of constraints. Thus the OT error-bound depends on the number n of constraints only through the quadratic factor (17a). In conclusion, the whole OT error-bound grows slowly (namely, quadratically) with the number of constraints. This means in turn that the OT learner is efficient also for realistically large constraint sets even in the worst-case scenario of the most adversarial training sequence. I thus conclude that the OT implementation of constraint-based phonology is well suited to support

error-driven learning: the OT framework is able to boost the limited computational resources of the error-driven learning scheme, making good theoretical guarantees nonetheless possible.

Let me now turn to the case of the HG error-driven leaner. Theorems 1 and 2 provide the explicit bounds (5) and (12) on the number of errors made by the HG error-driven learner with the original and the truncated Perceptron reweighing rule. These bounds are summarized in (18), which again consists of the product of two factors. The first factor (18a) is the square of the number n of constraints. The second factor (18b) involves the margin of either the training data (in the case of the original Perceptron) or the training plus the dummy data (in the case of the truncated Perceptron).

$$\text{(18)}\quad \text{number of errors made by the HG learner} \leq \underbrace{n^2}_{\text{(a)}} \times \underbrace{\frac{1}{\big(\text{margin of the data}\big)^2}}_{\text{(b)}}$$

The HG error-bound (18) looks superficially analogous to the OT error-bound (17). In both cases, the first factor (17a) and (18a) grows slowly (namely, quadratically) with the number of constraints. Yet, the two error-bounds are substantially different. In fact, the second factor (17b) in the OT bound does not depend on the number of constraints, as just noted. Instead, the margin which appears in the second factor (18b) of the HG error-bound crucially depends on the data and thus in particular on the number n of constraints. In other words, the HG error-bound depends on the number n of constraints through both factors (18a) and (18b). Because of this fact, although the first factor (18a) of the HG error-bound grows only quadratically in n, the entire error-bound can grow mush faster than quadratically. Indeed, consider a scenario where the (squared) margin decreases very fast (namely, exponentially). This means in turn that the second factor (18b) grows very fast (namely, exponentially), trumping the slow (namely, quadratic) growth of the first factor (18a). The overall error-bound thus grows very fast (namely, exponentially) and becomes astronomically large already for constraint sets of a realistic size. In this scenario, the error-bound (18) provides no guarantees of practical efficiency. Simulation results are needed in order to determine whether the fast growth of the error-bound is due to a looseness of the analysis (so that a tighter error-bound with a slower growth is possible) or whether instead the number of errors effectively made by the algorithm grows exponentially fast in the worst-case, thus matching the error bound.

In the rest of this section, I present a case which indeed illustrates the latter scenario. Indeed, the margin decreases very fast (namely, exponentially) in this counterexample, allowing the number of errors made by the HG error-driven learner (with both the original and the truncated Perceptron update rule) to grow very fast (namely, exponentially) with the number of constraints. So fast that over five million errors are made by the HG error-driven learner to learn the weights of just ten constraints (contrary to less than fifty errors made by the OT error-driven learner!). Crucially, the constraint set considered in this counterexample has the property that the corresponding typologies according to HG and OT exactly coincide. The inefficiency of the HG error-driven learner thus cannot be due to the fact that the algorithm is exploring a larger typology. In other words, even if the typologies predicted for a given constraint set 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 still outperforms the HG one from the perspective of error-driven learning.

To start, consider the case with $n = 5$ constraints C_1, C_2, C_3, C_4, and C_5. Assume there is a unique underlying form x which comes with five candidates, called y and z_1, z_2, z_3, and z_4. As usual, constraint violations are described by the number of stars in (19). The OT and HG typologies (computed with `OT-Help`; Staubs, Becker, Potts, Pratt, McCarthy, and Pater 2010) corresponding to this constraint set coincide: they both consist of five grammars, one for each mapping of the underlying form to one of the five candidates. In other words, the mode of constraint interaction has no typological effects in this case. Assume that the error-driven learner is trained on the target grammar which maps the underlying form x into the winning candidate y, which therefore beats the four other losing candidates ~~z_1~~, ~~z_2~~, ~~z_3~~, and ~~z_4~~.[5]

(19)

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		***	***	***	

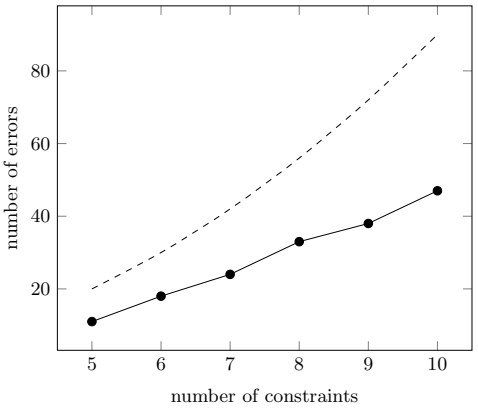
In order to study the growth of the number of errors made by error-driven learners as a function of the number n of constraints, the desired counterexample needs to contain a constraint set of cardinality n for each n. The extension from the case $n = 5$ to the case of an arbitrary number n of constraints is rather straightforward, once

[5]The constraints defined in (19) are not binary (the number of violations can be larger than one), contrary to what assumed throughout this paper in order to keep the notation simple. Thus, in the HG simulations reported below, I am using not the update condition (3) and the update rules (4)/(7) but their generalizations from binary to gradient constraints provided in appendix A.

the counterexample is described through *elementary ranking* and *weighting conditions* (Prince 2002; Magri 2013b), as explained in appendix B.1.

For each choice of the number n of constrains between 5 and 10, I have run the OT error-driven learner ten times on the corresponding counterexample; see appendix B.2 for simulation details and results. The solid line in (20) plots the largest number of errors[6] (vertical axis) performed by the algorithm over the ten runs as a function of the number n of constraints (horizontal axis). To get a sense of the scale of the increase of the number of errors with the number n of constraints, the dashed line in (20) plots the error-bound (17), which boils down to the function $n(n-1)$, as computed in appendix B.2.

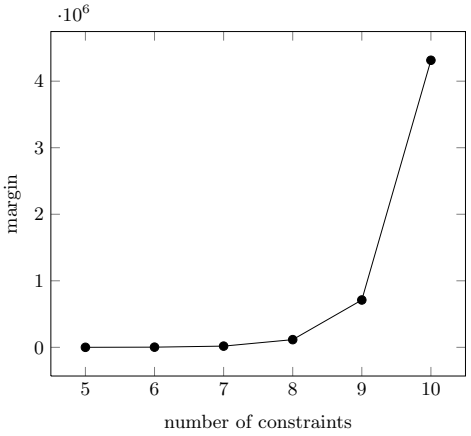
(20)



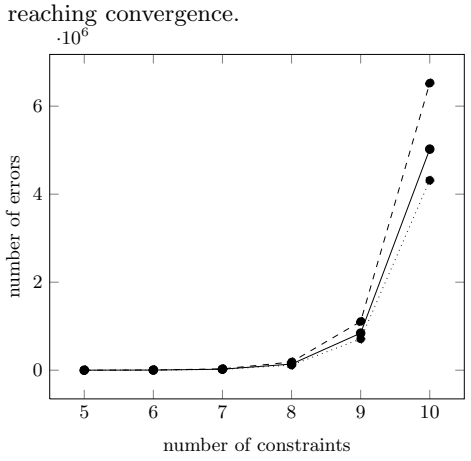
As guaranteed by the convergence theorem 3 for OT error-driven learning, we see in (20) that the number of errors grows slowly with the number of constraints, so slowly that less than fifty errors suffice to reach convergence in the case corresponding to $n = 10$ constraints.

The behavior of the HG error-driven learner is very different. The margin of the training data (as computed in appendix C.4) decreases very fast (namely, exponentially) with the number n of constraints. The squared inverse margin (18b) in the HG error-bound thus increases very fast, as plotted in (21a). The whole HG error-bound (18) thus becomes meaningless, as it would be compatible with the HG error-driven learner performing an astronomical number of errors. And indeed, that turns out to be the case. The solid and dashed lines in (21b) plot the largest number of errors made by the HG error-driven learner over ten runs, with the original Perceptron update rule (4) and the truncated variant (7), respectively.[7] These plots clearly shows that the number of errors made by the HG error-driven leaner grows exponentially with the number of constraints. Indeed, the case with just ten constraints already forces the algorithm to perform over five million errors before reaching convergence.

(21) a.



b.



This counterexample can be strengthened further, as follows. The HG error-bound (18) provides an *upper bound* on the *worst-case* number of errors: it guarantees that the number of errors can never be larger than a certain quantity, not even when trained on the worst, most adversarial training sequence. As explained in appendix C.5, in the case of the original Perceptron reweighing rule, it is possible to compute also a *lower bound* on the *best-case* number of errors: it guarantees that the number of errors needed to reach convergence can never be smaller than a certain quantity, not even when trained on the best, most favorable training sequence.[8] This lower-bound on the best-case number of errors is plotted with a dotted line in (21b), showing that even the best-case number of

[6] I plot the *largest* number of errors over the ten runs (rather than, say, the mean number of errors or the smallest number of errors) because the perspective adopted in this paper is that of error-bounds, namely the *worst-case* number of errors.

[7] Consistently with the worst-case perspective adopted in this paper, I plot in (21b) the *largest* number of errors made by the HG learner over the ten runs (see also footnote 6). But the plot of the *smallest* number of errors is almost indistinguishable.

[8] The technique used in appendix C.5 to compute a lower-bound on the best-case number of errors for the HG learner with the *original* Perceptron reweighing rule does not extend in any straightforward way to the *truncated* variant.

errors grows exponentially in the number of constraints. I conclude that the OT error-driven learner substantially outperforms HG error-driven learners such as the (original and truncated) Perceptron from the perspective of worst-case error-bounds.

3.4. **Discussion**

How does the conclusion reached at the end of the previous subsection 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 as an HG error-driven learner, Boersma and Pater (to appear) mention its successful application to large data sets in Computational Linguistics, as demonstrated for instance in 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) 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 [...]" (p. 1021).

It turns out that there is no contradiction at all between the conclusion reached at the end of the previous subsection 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 version of the Perceptron mentioned in the preceding quote from Mohri et al. (2012) and tested in Collins (2002). Unfortunately, the kernel Perceptron is not an *error-driven* algorithm. This variant of the Perceptron thus has nothing to do with the error-driven learning scheme investigated in this paper. The conclusions reached in the Machine Learning and Computational Linguistics literature concerning the kernel Perceptron are therefore irrelevant to the topic of this paper, namely the development of proper error-driven learners. In the rest of this subsection, I elaborate on this point.

The Perceptron convergence theorem 1 provides an error-bound for the Perceptron which depends on the *margin* of the training data, which can be interpreted as a measure of their "degree of consistency" with a target HG grammar. Unfortunately, it is easy to construct cases where the margin of the training data goes to zero very fast with n, as in the counterexample constructed in subsection 3.3. In these cases, the number of errors made by the Perceptron can — and indeed does — grow very fast. Indeed, the Machine Learning and Computational Linguistics literature is concerned with applications where the margin of the training data is not only small but actually "negative", as the data usually fail to be consistent. A state-of-the-art solution to this problem is provided by the *kernel method* (Schölkopf and Smola 2002). The core idea is to run the learning algorithm not on the original training data of dimension n but rather on derived data obtained by embedding the original data into a space of a much higher (possibly infinite) dimension N. The advantage is that increasing the dimension boosts the margin: a training set which is inconsistent or has a small margin in the original n-dimensional space usually yields derived data with a large margin in a space of larger dimension N. The obvious disadvantage is that increasing the dimension worsens the computational burden: in general, it will take longer to perform a certain operation in a space of larger dimension N than in the original space of smaller dimension n.

Quite surprisingly, it turns out that there are some cases where you have your cake and it too. Indeed, if the derived N-dimensional space is linked to the original n-dimensional space through a so called *kernel function*, then some algorithmic operations on the derived N-dimensional training data can be easily computed just in terms of the original n-dimensional data. We are thus interested in algorithms that only perform operations which have this special property. Such algorithms are called *kernelizable*. In other words, an algorithm is kernelizable if it can be run on a high-dimensional space boosting the margin of the data without losing anything in terms of computational efficiency. Unfortunately, the Perceptron is not kernelizable. Yet, it turns out that the Perceptron admits a *dual* formulation which is equivalent to the original *primal* Perceptron, in the sense that the two algorithms walk through the same sequence of weight vectors and learn the same final vector when trained on the same sequence of data. And crucially, this dual variant of the Perceptron turns out to be kernelizable. It is for this reason that the Machine Learning and Computational Linguistics literature has indeed focused on this kernel dual Perceptron.

What allows the dual, but not the primal, Perceptron to undergo kernelization is a richer representation of its past learning history. The primal Perceptron maintains an extremely impoverished summary of its past learning history. In fact, it does not keep track of the updates that have been previously performed nor of the training data that triggered those updates. The only memory of that past learning history consists of the way those past updates have shaped the current weight vector. In order to allow kernelization, the dual Perceptron instead maintains a richer history of its past experience. Indeed, it needs to store the (finite) set of training data and for each piece of training data it needs to maintain a counter (called a *dual variable*) of the number of times that piece of data has triggered an update up to that time in the run. We can summarize this difference between the primal and the dual Perceptron by saying that only the former, but not the latter, qualifies as an *error-driven* learning algorithm. This difference between the two implementations is of course irrelevant from the engineering perspective of the Machine Learning and Computational Linguistics literature.

Unfortunately, this difference between the primal and the dual implementation of the Perceptron does have a crucial importance from a cognitive modeling perspective. In fact, certain aspects of the target phonology are plausibly acquired before the development of the native language lexicon. The early acquisition of the native

phonotactics is one such paradigmatic case (Hayes 2004). In order to model these early acquisition stages, we need special learning algorithms that do not need to store and act upon the entire lexicon. Error-driven learning algorithms stand out as particularly suited to this task. For instance, OT error-driven ranking algorithms and the HG error-driven Perceptron algorithm described above do not need any access to the lexicon, as the updates are performed instantaneously based on the current piece of data and they do not keep track of the updates that have been previously performed nor of the training data that triggered those updates. The case of the dual Perceptron is different, as it needs access to the entire lexicon in order to index its dual variables. The dual Perceptron therefore is not an error-driven algorithm and does not qualify as a proper model of the early stages of language acquisition, namely those stages which happens before the development of the native language lexicon.

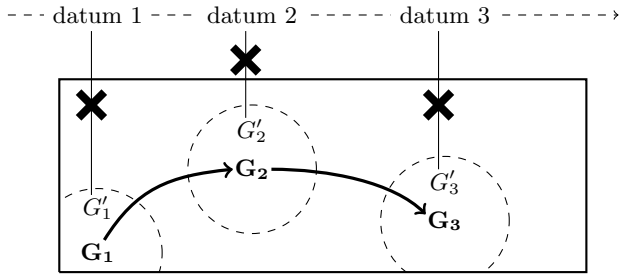
3.5. **Summary**

The informal error-driven learning scheme (1) can be implemented within a variety of grammatical frameworks. In this section, I have compared the OT and HG implementations from the perspective of the following question: what is the number of errors made in the worst possible case, namely when trained on the most adversarial training sequence? The answer to this question developed in this section has three parts. *First*, I have noted that the error-bound for the OT implementation outperforms the error-bound for the HG implementation (with both the original and truncated Perceptron reweighing rules), as only the OT error-bound grows slowly (namely, quadratically) in the number of constraints. *Second*, I have reported simulation results 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. *Third*, I have argued that the poor performance of the HG learner with the original and truncated Perceptron reweighing rule does not contradict the good performance of the Perceptron reported in the Machine Learning and Computational Linguistics literature, as the latter literature has looked at an implementations of the Perceptron which is not *error-driven*, and thus does not bear on the issue of the proper implementation of error-driven learning addressed in this paper.

4. Stochastic error-driven learning is much slower in HG than in OT

The implementation of error-driven learning schematized in (1) is called *deterministic*, to distinguish it from the *stochastic* implementation schematized in (22). 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 of that current grammar (say, G'_1) which is sampled at random (according to a certain probability distribution defined on the space of grammars) from a small neighborhood centered around the current grammar (depicted here as a dashed circle centered at G_1).

(22)



Various authors have explored the modeling implications of stochastic error-driven learning for modeling language variation and gradient acceptability judgments (Boersma 1997, 1998; Boersma and Hayes 2001; Coetzee and Pater 2008, 2011; Coetzee and Kawahara 2013; Boersma and Pater to appear). In this subsection, I thus compare the OT and HG implementations of this stochastic variant of error-driven learning.

4.1. **Stochastic error-driven learning in HG and OT**

The informal stochastic error-driven learning scheme (22) can be formalized as follows in HG and OT. According to the original deterministic implementation (2), the error-driven learner maintains a current grammar represented through a (weight or ranking) vector $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$ and checks at step (2c) whether this current grammar needs to be updated or whether instead it already manages to make the current intended winner beat the current intended loser. In the case of HG (under the assumption of binary constraints), this update condition boils down to (3), repeated in (23a). In the case of OT, the update condition boils down to (13), repeated in (23b).

(23) a. HG deterministic update condition:
$$\sum_{h \in W} \theta_h - \sum_{k \in L} \theta_k > 0$$

b. OT deterministic update condition:
$$\max_{h \in W} \theta_h - \max_{k \in L} \theta_k > 0$$

The only innovation of the *stochastic* variant of error-driven learning is that at step (2c) it samples a *stochastic vector* $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)$, namely a collection of n numbers $\epsilon_1, \ldots, \epsilon_n$ drawn independently from each other according to a certain underlying probability distribution. It adds component-wise this stochastic vector $\boldsymbol{\epsilon}$ to the current (weight or ranking) vector $\boldsymbol{\theta}$, obtaining the vector $\boldsymbol{\theta} + \boldsymbol{\epsilon} = (\theta_1 + \epsilon_1, \ldots, \theta_n + \epsilon_n)$. And it checks the update condition not for the current grammar corresponding to the current (weight or ranking) vector $\boldsymbol{\theta}$, but for the stochastic

variant of that grammar which corresponds to the (weight or ranking) vector $\boldsymbol{\theta} + \boldsymbol{\epsilon}$. In other words, the algorithm checks the update conditions (24), which are the original conditions (23) applied to $\boldsymbol{\theta} + \boldsymbol{\epsilon}$ rather than to $\boldsymbol{\theta}$.

(24) a. HG stochastic update condition:
$$\sum_{h \in W} (\theta_h + \epsilon_h) - \sum_{k \in L} (\theta_k + \epsilon_k) > 0$$

b. OT stochastic update condition:
$$\max_{h \in W} (\theta_h + \epsilon_h) - \max_{k \in L} (\theta_k + \epsilon_k) > 0$$

By rearranging the stochastic numbers on the right hand side of the inequalities, conditions (24) can be brought into the shape (25). The value ϵ_{HG} which appears on the right hand side of (25a) is the sum of those stochastic values $\epsilon_1, \dots, \epsilon_n$ which correspond to loser-preferring constraints minus the sum of those stochastic values which correspond to winner-preferring constraints. The value ϵ_{OT} which appears on the right hand side of (25b) is the difference between the stochastic value corresponding to the top loser-preferring constraint minus the stochastic value corresponding to the top winner-preferring constraint.[9]

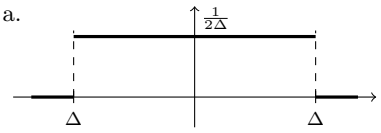
(25) a. HG stochastic update condition:
$$\sum_{h \in W} \theta_h - \sum_{k \in L} \theta_k > \epsilon_{\text{HG}}$$

b. OT stochastic update condition:
$$\max_{h \in W} \theta_h - \max_{k \in L} \theta_k > \epsilon_{\text{OT}}$$

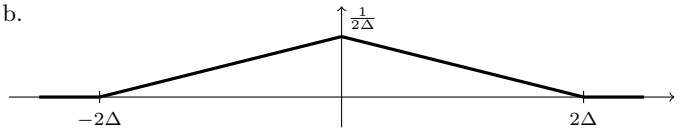
The algorithm (2) with the update conditions (24)/(25) at step (2c) is called the *stochastic* error-driven learner.

To complete the description of HG and OT stochastic error-driven learning, we need to specify the probability distribution used to sample the stochastic vector $\boldsymbol{\epsilon} = (\epsilon_1, \dots, \epsilon_n)$. Starting with seminal work by Boersma (1997, 1998), the literature has used a gaussian distribution with zero mean and a small variance σ. Since the normal distribution is symmetric, the stochastic value ϵ_k and its opposite $-\epsilon_k$ have the same probability. Furthermore, since the tails of the normal distribution decrease exponentially fast, the stochastic value ϵ_k is bounded between $-\Delta$ and Δ with high probability (which of course depends on Δ). For instance, if $\Delta = 4\sigma\sqrt{2}$, then the probability that the stochastic value is bounded within $-\Delta$ and Δ is 1.00000. From an analytical perspective, it is nonetheless convenient to make the stochastic values deterministically bounded, rather than bounded with high probability. One way to achieve that is to replace the gaussian distribution with a uniform distribution between $-\Delta$ and Δ, plotted in (26a). If $\Delta = 0$, then all the numbers ϵ_k are equal to 0 and the stochastic implementation of error-driven learning thus coincides with the deterministic implementation. In other words, the constant Δ measures the "size" of the stochastic component of the stochastic implementation.

(26) a.



b.



Even though the uniform and the gaussian distribution are quite different, the choice between the two can hardly have any effect on the behavior of the algorithm. In fact, the stochastic values $\epsilon_1, \dots, \epsilon_n$ enter into the HG and OT stochastic update conditions (25) never by themselves but only through the sum/difference of at least two of them. And the sum/difference of two uniform random variables has the triangular distribution plotted in (26b), which is quite close to a gaussian distribution. To keep the reasoning simple in the rest of this subsection, I assume that the stochastic values $\epsilon_1, \dots, \epsilon_n$ are indeed sampled according to the uniform distribution between $-\Delta$ and Δ, and therefore can never be smaller than $-\Delta$ nor larger than Δ. The reasoning carries over with high probability to the case where the stochastic values are sampled according to the gaussian distribution.

4.2. **Error-bounds for stochastic error-driven learning in HG and OT**

Since the stochastic values ϵ_k cannot be larger than Δ, then the value ϵ_{HG} which appears on the right hand side of the stochastic HG update condition (25a) is bounded as well (and the bound depends on Δ). A comparison between the deterministic HG update condition (23a) and the stochastic condition (25a) reveals that they are basically identical, apart from the fact that zero on the right hand side of the former has been replaced in the latter by a certain value ϵ_{HG} which could be different from zero but still not too large. Since the deterministic and stochastic implementations only differ for the update conditions and since the update conditions differ only minimally, the convergence theorems 1 and 2 for the (original and truncated) deterministic Perceptron extend straightforwardly to the (original and truncated) stochastic variant, as first noted in Boersma and Pater (to appear). These considerations yield the following convergence theorem 4; see appendix C.3 for details.

Theorem 4. *The stochastic HG error-driven learner (2) with the stochastic update condition (25a) and the (original or truncated) Perceptron update rule (4)/(7) converges: whenever trained on data consistent with an HG grammar, it can only perform a finite number of errors, before converging to final weights which are guaranteed to be consistent with the target grammar, so that no further updates are triggered and learning ceases. Furthermore, the number of errors made before converging can be bound as follows:*

[9] More explicitly: $\epsilon_{\text{HG}} = \sum_{k \in L} \epsilon_k - \sum_{h \in W} \epsilon_h$ and $\epsilon_{\text{OT}} = \epsilon_{k*} - \epsilon_{h*}$ where $\theta_{k*} + \epsilon_{k*} = \max_{k \in L} \theta_k + \epsilon_k$ and $\theta_{h*} + \epsilon_{h*} = \max_{h \in W} \theta_h + \epsilon_h$.

$$\text{(27)}\quad \textit{number of errors} \;\leq\; \underbrace{\frac{n^2}{\big(\textit{margin of the data}\big)^2}}_{\text{(a)}} + \underbrace{\frac{2\Delta n}{\big(\textit{margin of the data}\big)^2}}_{\text{(b)}}$$

in terms of the number n of constraints, a measure Δ of the size of the stochastic component of the algorithm, and the margin of either the training data (for the original Perceptron reweighing rule) or the training plus the dummy data (for the truncated Perceptron reweighing rule). ■

Analogous considerations hold of course for the OT implementation of stochastic error-driven learning. Again, a comparison between the deterministic OT update condition (23b) and the stochastic condition (25b) reveals that they are basically identical, apart from the fact that zero on the right hand side of the former has been replaced in the latter by a certain value ϵ_{OT} which could be different from zero but still not too large. Since the deterministic and stochastic implementations only differ for the update conditions and since the update conditions differ only minimally, the convergence theorem 3 for deterministic calibrated error-driven ranking algorithms extends straightforwardly to the stochastic variant, yielding the following theorem 5; see Magri (2013a) for details. For the case of demotion-only error-driven ranking algorithms (that is, algorithms that adopt the re-ranking rule (14) with a null promotion amount $p = 0$), the error-bound (28) is tight, namely cannot be improved.

Theorem 5. *The stochastic OT error-driven learner (2) with the stochastic update condition (25b) and a calibrated update rule (14) converges: whenever trained on data consistent with an OT grammar, it can only perform a finite number of errors, before converging to a final ranking vector. Any constraint ranking which is consistent with these final ranking values (namely, which satisfies the ranking conditions induced by their relative size) is consistent with the target grammar. Furthermore, the number of errors made before converging can be bound as follows:*

$$\text{(28)}\quad \textit{number of errors} \;\leq\; \underbrace{\frac{n^2}{\textit{calibration}}}_{\text{(a)}} + \underbrace{\frac{2\Delta n^2}{\textit{calibration}}}_{\text{(b)}}$$

in terms of the number n of constraints, a measure Δ of the size of stochastic component of the algorithm, and the calibration of the promotion amount used in the re-ranking rule. ■

4.3. **Comparison**

In the worst-case scenario, we expect that the stochastic component can derail the error-driven leaner away from the most straightforward path to success. In other words, we expect a stochastic error-driven learner to make more errors and to converge more slowly than the corresponding deterministic learner. This additional number of errors quantifies the sensitivity of the error-driven learner to the stochastic component. The fundamental questions addressed by the theory of stochastic error-driven learning in constraint-based phonology can thus be informally stated as follows: how large is the number of additional errors caused by the stochastic component? Does the size of this slowdown depend on the choice between the HG and OT implementation of stochastic error-driven learning? In that case, which of the two implementations makes the smallest number of additional errors and is thus least affected by the stochastic component?

Here is a way to state state these questions explicitly. Let's express the number of errors made by a stochastic error-driven learner as the sum of two terms: the first term (29a) is the number of errors made by the corresponding deterministic learner, which measures the intrinsic difficulty of error-driven learning; the second term (29b) quantifies the number of additional errors due to the stochastic component, measuring the slowdown due to the stochastic component.

$$\text{(29)}\quad \begin{array}{c}\text{number of errors made by}\\ \text{the } \textit{stochastic} \text{ learner}\end{array} \;=\; \underbrace{\begin{array}{c}\text{number of errors made by}\\ \text{the } \textit{deterministic} \text{ learner}\end{array}}_{\text{(a)}} + \underbrace{\begin{array}{c}\text{additional term due to}\\ \text{the stochastic component}\end{array}}_{\text{(b)}}$$

In the preceding section 3, we have focused on the deterministic error-bound (29a). In this section, we carry out the same analysis for the additional term (29b) due to the stochastic component. Of course, the number (29b) of additional errors due to the stochastic component should be allowed to grow with the complexity of the learning task. In the case of constraint-based phonology, the simplest measure of the complexity of the learning task is plausibly the number of constraints that the learner needs to rerank or reweigh. In principle, two scenarios might arise. According to one scenario, the number (29b) of additional errors due to the stochastic component grows as a function of the number of constraints but only slowly, so that this additional number of errors remains relatively small also for realistically large constraint sets. According to the opposite scenario, this number (29b) of additional errors grows instead very fast (namely, exponentially) with the number of constraints, so that it is already astronomically large for realistic constraint sets.[10] The fundamental questions of the theory of error-driven learning in constraint-based phonology can thus be formalized as follows: how fast does this number (29b)

[10] Of course, we expect the number (29b) of additional errors due to the stochastic component to also depend on the size Δ of the stochastic component itself. If Δ is null, the stochastic component is switched off, because the stochastic vector $\boldsymbol{\epsilon}$ has null components, so that the current vector $\boldsymbol{\theta}$ and its stochastic counterpart $\boldsymbol{\theta} + \boldsymbol{\epsilon}$ coincide. In this case, we thus expect the additional

of additional errors grow with the number of constraints in the case of the OT and the HG implementation of stochastic error-driven learning? which of the two implementations yields the slowest growth and is thus least affected by the stochastic component?

Let me start with the case of the OT stochastic error-driven learner. Theorem 5 provides an explicit bound (28) on the number of errors that the algorithm can make, even in the worst-case scenario of the most adversarial training sequence. The two terms of this bound (28) correspond to the two terms of the general scheme (29). In fact, the first term (28a) coincides with the error-bound (17) obtained for the deterministic OT error-driven learner. The second term (28b), repeated in (30), then quantifies the additional number of errors due to the stochastic implementation.

$$\text{(30)}\qquad \begin{array}{c}\text{number of } \textit{additional} \text{ errors}\\ \text{made by the stochastic OT learner}\end{array} \leq \underbrace{2\Delta}_{\text{(a)}} \times \underbrace{n^2}_{\text{(b)}} \times \underbrace{\frac{1}{\text{calibration of the promotion amount}}}_{\text{(c)}}$$

The number (30) of additional errors due to the stochastic component is the product of three factors. The first factor (30a) captures the dependence of the additional number of errors on the size Δ of the stochastic component. The second factor (30b) depends quadratically on the number n of constraints. Finally, the third factor (30c) depends on the calibration constant. Crucially, the calibration constant does not depend on the number n of constraints. Thus the overall bound (30) on the number of additional errors due to the stochastic component depends on the number n of constraints only through the quadratic factor (30b). In other words, the number of additional errors grows slowly (quadratically) with the number of constraints. This fact has two implications. First, that the number of additional errors does not blow up for realistically large constraint sets. Second, that the entire error bound (28) for the *stochastic* implementation of OT error-driven learning has the same quadratic dependence on the number of constraints as the error-bound (15) for the *deterministic* implementation. In other words, the stochastic component has no effect on the speed of growth of the overall number of errors with the number of constraints. I thus conclude that the OT implementation of constraint-based phonology is well suited to support stochastic error-driven learning, as the stochastic component causes an increase in the number of errors which grows only slowly with the number of constraints and does not affect the speed of growth of the overall number of errors.

Let me now turn to the case of the HG error-driven leaner. Theorem 4 provides an explicit bound (27) on the number of errors that the algorithm can make, even in the worst-case scenario of the most adversarial training sequence. Also the two terms of this error-bound (27) correspond to the two terms of the general scheme (29). In fact, the first term (27a) coincides with the error-bound (18) obtained for the deterministic HG learner. The second term (27b), repeated in (31), then quantifies the additional number of errors due to the stochastic implementation.

$$\text{(31)}\qquad \begin{array}{c}\text{number of } \textit{additional} \text{ errors}\\ \text{made by the stochastic HG learner}\end{array} \leq \underbrace{2\Delta}_{\text{(a)}} \times \underbrace{n}_{\text{(b)}} \times \underbrace{\frac{1}{\big(\text{margin of the data}\big)^2}}_{\text{(c)}}$$

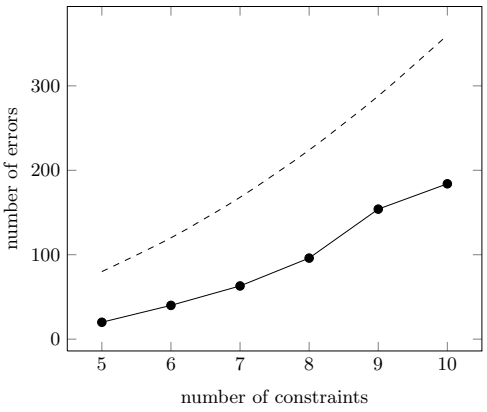
The number (31) of additional errors due to the stochastic component is the product of three factors. The first factor (31a) captures the dependence of the additional number of errors on the size Δ of the stochastic component. The second factor (31b) depends linearly on the number n of constraints. Finally, the third factor (31c) depends on the margin of the training data (in the case of the original Perceptron reweighing rule) or the training plus dummy data (in the case of the truncated Perceptron reweighing rule). The bound (31) on the additional number of errors made by the stochastic HG learner looks superficially analogous to the bound (30) for the stochastic OT learner. Yet, the two bounds are substantially different, in exactly the same way discussed above in subsection 3.3. In fact, the margin which appears in the factor (31c) of the HG bound does depend on the data and thus also on the number n of constraints. In other words, the bound (31) depends on the number n of constraints through both factors (31b) and (31c). Because of this fact, although the former factor (31b) grows slowly (namely, linearly) in n, the entire bound could grow much faster. Indeed, consider a scenario where the (squared) margin decreases very fast (namely, exponentially). This means in turn that the factor (31c) grows very fast (namely, exponentially), trumping the slow (namely, linear) growth of the factor (31b). The overall bound (31) thus grows very fast (namely, exponentially) and becomes astronomically large already for constraint sets of a realistic size. In this scenario, the bound on the number of additional errors made by the HG stochastic error-driven learning provides no guarantees on the sensitivity of the error-driven learner to the stochastic component. Simulation results are needed in order to determine whether the fast growth of the bound is due to a looseness of the analysis (while a tighter bound, with a slower growth is possible) or whether instead the number of additional errors effectively made by the algorithm grows exponentially fast in the worst-case, thus matching the error bound.

---

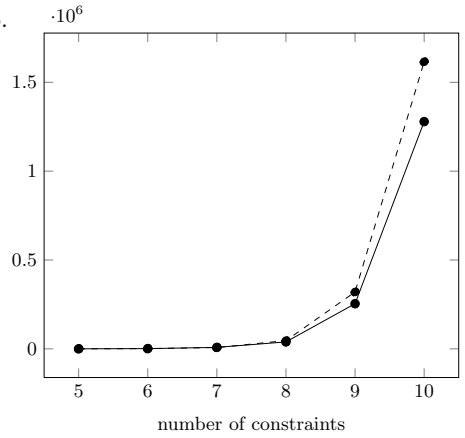
term (29b) to be null. As Δ grows larger than zero, the stochastic component becomes more relevant, because the stochastic vector $\boldsymbol{\epsilon}$ is allowed to have larger components, so that the current vector $\boldsymbol{\theta}$ and its stochastic counterpart $\boldsymbol{\theta} + \boldsymbol{\epsilon}$ get further apart. In other words, as Δ grows larger than zero, the additional term (29b) is expected to grow as well. It turns out that it grows slowly (namely, linearly) in Δ in both the OT and the HG implementations, as shown in the additional term (27b) for the HG implementation and the additional term (28b) for the OT implementation. As the rate of growth with respect to Δ does not distinguish between the two implementations, I ignore it in the rest of this section.

Indeed, the counterexample constructed in section 3.3 can be used to show that the additional number of errors due to the stochastic implementation of the HG error-driven learner (with both the original and the truncated Perceptron update rule) can grow very fast (namely, exponentially) with the number of constraints. Here are the details. For each choice of the number n of constrains between 5 and 10, I have run the deterministic and stochastic OT error-driven learner ten times on the corresponding counterexample; see appendix B.2 for simulation details. I plot in (32a) with a solid line the difference between the largest number of errors made by the stochastic algorithm over the ten runs minus the number of errors made in that same run by the deterministic algorithm. In other words, the graph plots the additional number of errors due to the stochastic component (vertical axis) as a function of the number n of constraints (horizontal axis). To get a sense of the scale of the increase of the number of errors with the number n of constraints, I plot with a dashed line the term (28b) in theorem 5, which boils down to the function $4n(n-1)$, as computed in appendix B.2. As guaranteed by theorem 5, we see in (32a) that the difference between the number of errors made by the stochastic and the deterministic OT error-driven learner grows slowly with the number of constraints, so that it is smaller than 200 in the case corresponding to $n = 10$ constraints.

(32) a.



b.



In (32b) I turn to the HG error-driven learner; see appendix B for simulation details and results. The solid and dashed lines plot the difference between the largest number of errors made by the stochastic learner over the ten runs minus the number of errors made in that same run by the corresponding deterministic learner in the case of the original Perceptron update rule (4) and the truncated variant (7), respectively. The graph clearly shows that the number of additional errors due to the stochastic component grows exponentially with the number of constraints. Indeed, the case with just ten constraints already forces the stochastic learner to perform over one million additional errors compared with the deterministic learner, which in turn was already performing over five million errors! I conclude that the OT implementation of stochastic error-driven learning substantially outperforms the HG implementation from the perspective of worst-case error-bounds.

4.4. **Summary**

Section 3 has compared the OT and HG *deterministic* error-driven learners from the perspective of error-bounds. This section has carried this comparison further, comparing the OT and HG *stochastic* error-driven learners from the perspective of the following question: how worse is the error-bound for the stochastic learner compared with the error-bound for the corresponding deterministic learner? The answer to this question developed in this section has two parts. *First*, I have noted that the error-bound for the stochastic OT leaner outperforms the error-bound for the HG learner, as only the OT bound on the additional number of errors grows slowly (namely, quadratically) in the number of constraints. *Second*, I have reported simulation results where the HG and OT typologies coincide and yet the additional number of errors made by the HG learner because of the stochastic component (namely, discounting the errors made by the deterministic learner on that same training sequence) is over one million with just ten constraints, which is well over 5,000 times the number of additional errors made by the OT learner! The conclusions reached in this section thus align with those reached in the previous section 3, both showing a clear superiority of OT over HG error-driven learning.

5. Conclusions

Two main implementations of constraint based phonology have been pursued in the literature: HG, which assumes a weighted model of constraint interaction; and OT, which assumes instead the principle of strict domination. One of the main open architectural issues in constraint-based phonology thus concerns the choice between these two implementations. At first sight, this might look like an exquisitely typological issue: the choice between HG and OT should depend on their match to the actual typology of (attested and allegedly possible) natural language phonologies. Unfortunately, the problem of determining the proper mode of constraint interaction is unlikely to be settled on a typological basis, as various authors have acknowledged (Pater 2009, Bane and Riggle 2010). The difficulty is that we have to deal at the same time with two unknowns, namely both the mode of constraint interaction and the proper definition of the phonological constraints themselves. If we happened to know what the

actual constraint set looks like, then we could easily compute the typological predictions made by the two modes of constraint interaction and determine which one yields the most accurate typological predictions. On the other hand, if we happened to know the actual mode of constraint interaction, we could likely reconstruct the constraint set from typological data. The problem is that we don't know neither the constraints nor their mode of interaction. And these two unknowns are coupled, as different modes of constraint interaction might require different constraint sets. In conclusion, *descriptive adequacy* is not likely to pull apart the OT and the HG implementations of constraint-based phonology.

It is for this reason that the research trying to probe into the relative merits of the two frameworks has turned from descriptive to *explanatory adequacy*. The idea is to evaluate the two frameworks apart from their typological predictions, by distilling their algorithmic implications for the development of proper computational models of the production, perception, and acquisition of phonology. For instance, Riggle (2009) and Bane et al. (2010) compare the two frameworks from the perspective of learning-theoretic complexity measures such as their *VC-dimension*; Magri (2013b) compares them from the perspective of the computational problem of *consistency*, namely the problem of finding a grammar consistent with a finite number of data; Jesney and Tessier (2011) compare them from the perspective of the problem of *restrictiveness* in phonotactic learning, namely the problem of learning which forms are phonotactically illicit while being trained on licit forms only. This paper contributes to this enterprise, as it provides new elements to compare the two frameworks of OT and HG from the algorithmic perspective of error-driven learning.

Certain aspects of the native language phonology (such as the native phonotactics) are acquired prior to the development of the native language lexicon. An error-driven learner is ideally suited to model these early acquisition stages, as it performs instantaneous updates, based on a single piece of data at the time, and thus does not need to access and act upon the entire phonological lexicon. This modeling advantage turns of course into a computation liability: the fact that each update is computed instantaneously based only on the current piece of data means that the algorithm is unable to compute the implications of its current update for the rest of the dataset. The current update might thus later turn out to represent an error which needs to be laboriously corrected. Because of this intrinsic weakness of its computational means, error-driven learning can only succeed at the learning task when implemented within a very carefully crafted grammatical framework, which is able to boost the algorithmic strength of this learning scheme. Which of the two implementations of constraint-based phonology — OT or HG — is best suited to this task? An answer to this question might provide new elements in order to pull apart the two frameworks from the perspective of their algorithmic implications. This paper has thus addressed this question, though two main contributions which can be summarized as follows.

OT has been endowed with proper error-driven learners since the seminal work of Tesar and Smolensky (1998). The case of HG turned out to be more delicate. The HG literature has so far relied on error-driven algorithms for linear classification, and in particular on the Perceptron algorithm. Unfortunately, the Perceptron algorithm as it stands does not really qualify as an error-driven learner for HG. The problem is that HG crucially requires the constraint weights to be non-negative while the Perceptron does not in any way enforce non-negativity of the current and final weights. Heuristic strategies adopted so far in the literature to overcome this problem (for instance, starting from large initial weights) come with no guarantees of success (for instance, large initial weights in no way guarantee nonnegative final weights, as the number of updates depends on the size of the initial weights). The first contribution of this paper is a principled solution to this impasse. I have considered a variant of the original Perceptron which ensures non-negativity of the weights by truncating any update at zero. I have pointed out that any run of this truncated Perceptron can be mimicked through a run of the original Perceptron on an extended, consistent training set. And I have thus concluded that convergence guarantees and error-bounds for the original Perceptron extend to the truncated variant (theorem 2).

Now that we know that both OT and HG can be endowed with proper error-driven learners, we can ask the question of which one of the two frameworks is better suited to support error-driven learning. In this paper, I have thus compared error-driven learning in OT and HG from the perspective of how fast their worst-case error-bounds grow as a function of the number of constraints. The second contribution of this paper has been to show that OT substantially outperforms HG from this perspective. In fact, while the error-bound for the OT error-driven learner grows quadratically in the number of constraints (theorem 3), the counterexample constructed in section 3.3 shows that the number of errors made by the HG error-driven learner can grow very fast (exponentially) in the number of constraints. In other words, it is possible to construct cases where the number of errors made by the HG learner grows so fast with the number of constraints that the algorithm becomes unfeasible for any reasonable number of constraints. Crucially, the slowness of the HG learner compared to the OT one is not due to the fact that the former is exploring a larger typological space, as the counterexample constructed has the property that the two typologies predicted by OT and HG coincide. Finally, the superiority of the OT over the HG implementation holds no matter whether we consider the deterministic or the stochastic implementation of error-driven learning, as shown in section 4.

Assessing the relative merits of the OT and HG versions of constraint-based phonology from the perspective of their algorithmic implications is an enterprise still in its infancy. Even from the limited perspective of error-driven learning, there is of course much more than the worst-case number of errors. Furthermore, it could very well turn out to be the case that those counterexamples where the HG error-driven learner makes an astronomical number of errors (such as the case constructed in subsection 3.3) all turn out to be phonologically unrealistic. In other words,

efficiency guarantees for HG error-driven learning might be possible after all, by refining the theory of error-bounds through proper assumptions on what counts as a phonologically plausible constraint set. Nonetheless, the point sticks that more work is needed in order to develop efficiency guarantees for HG over OT error-driven learning, as only HG efficiency guarantees, but not OT guarantees, need to take into account carefully stated typological restrictions. In so far as OT allows for a more straightforward theory, it is better suited to develop error-driven learning and to rip off the modeling advantages of this learning scheme.

Appendix A. HG error-driven learning for general (not necessarily binary) constraints

To keep the notation lean, in the paper I have assumed that the constraints are binary, namely that they can assign at most one violation. In this appendix, I relax this restrictive assumption and provide the general description of the learning algorithms (subsections A.1 and A.2) and the general formulation of their theoretical guarantees (subsections A.3 and A.4) for the case where the constraints can assign an arbitrary number of violations.

A.1. **Description of the learning algorithms**

At steps (2a)-(2b), the HG error-driven learner receives the piece of information that a certain winner candidate y beats a certain loser candidate z relatively to a certain underlying form x according to the target HG grammar the algorithm is being trained on. At step (2c), the algorithm checks whether its current weights $\boldsymbol{\theta} = (\theta_1, \dots, \theta_n)$ already ensure that the winner y beats the loser z. In the general case of possibly non-binary constraints, the latter condition boils down to the strict inequality (33a), which says that the loser z violates the constraints more severely than the winner y according to the weights $\boldsymbol{\theta}$, in the sense that the weighted sum of the violations incurred by the loser (left hand side) is larger than the weighted sum of the violations incurred by the winner (right hand side).

(33) a. $\displaystyle\sum_{h=1}^{n} \theta_h C_h(x, \text{z}) > \sum_{k=1}^{n} \theta_k C_k(x, y)$

b. $\displaystyle\sum_{h=1}^{n} (\theta_h + \epsilon_h) C_h(x, \text{z}) > \sum_{k=1}^{n} (\theta_k + \epsilon_k) C_k(x, y)$

In the stochastic implementation of HG error-driven learning, the algorithm samples the components of a stochastic vector $\boldsymbol{\epsilon} = (\epsilon_1, \dots, \epsilon_n)$ independently according to the same underlying distribution and then checks the update condition (33a) not for the current weight vector $\boldsymbol{\theta} = (\theta_1, \dots, \theta_n)$ but rather for the perturbed weight vector $\boldsymbol{\theta} + \boldsymbol{\epsilon} = (\theta_1 + \epsilon_1, \dots, \theta_n + \epsilon_n)$, yielding condition (33b). In the case of binary constraints, the deterministic update condition (33a) and the stochastic condition (33b) indeed boil down to to the update conditions (3) and (24) considered in the paper.

If conditions (33) are satisfied, then the algorithm has nothing to learn from the current piece of data. Otherwise, the algorithm slightly updates its current weights at step (2d), using either the original or the truncated Perceptron reweighing rule. In the general case of possibly non-binary constraints, the original Perceptron reweighing rule takes the shape in (34a). Recall that constraint C_k is winner-preferring (loser-preferring) provided the number of violations $C_k(x, \text{z})$ it assigns to the intended loser z is larger (smaller, respectively) then the number of violations $C_k(x, y)$ it assigns to the intended winner y, in which case the quantity $C_k(x, \text{z}) - C_k(x, y)$ is positive (negative) and the weight of that constraint is therefore increased (decreased, respectively) by the update (34a).

(34) a. For each constraint C_k, add the quantity $C_k(x, \text{z}) - C_k(x, y)$ to its current weight θ_k.

b. For each constraint C_k, add the quantity $C_k(x, \text{z}) - C_k(x, y)$ to its current weight θ_k...
... unless that update would make that weight negative, in which case set that weight to zero

The truncated Perceptron reweighing rule (34b) coincides with the original Perceptron reweighing rule, apart from the additional "unless" clause in italics, meant to prevent the weights from ever turning negative. In the case of binary constraints, the original and truncated reweighing rules (34a) and (34b) indeed boil down to the original and truncated Perceptron reweighing rules (4) and (7) considered in the paper.

A.2. **Reformulation in EWC notation**

At steps (2a)-(2b), the HG error-driven learner receives the piece of information that a certain winner candidate y beats a certain loser candidate z relatively to a certain underlying form x. In this subsection, I show that this piece of information can be represented compactly, leading to a substantial simplification in the description of HG error-driven learners. Consider the n-tuple $\mathbf{a} = (a_1, \dots, a_n)$ whose kth entry a_k consists of the difference between the number of violations assigned by constraint C_k to the loser z minus the number of violations assigned to the winner y, as defined in (35). This n-tuple $\mathbf{a}$ is called an *elementary weighting condition* (EWC; Magri 2013b); for an example, see below (46) in appendix B.1. I will denote by $\mathbf{A}$ a finite collection of EWCs.

(35) $a_k = C_k(x, \text{z}) - C_k(x, y)$

Using the EWC notation, the deterministic update condition (33a) can be rewritten as in (36a), which requires the sum of the components a_k of the EWC weighted by the weights θ_k to be strictly positive. Furthermore, the stochastic update condition (33b) can be rewritten as in (36b), again using the perturbed weights $\theta_k + \epsilon_k$ in place of the original weights θ_k.

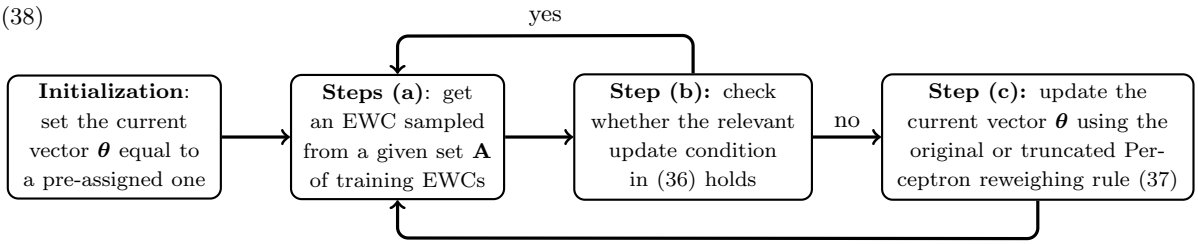
(36) a. $\sum_{k=1}^{n} a_k\theta_k > 0$ b. $\sum_{k=1}^{n} a_k(\theta_k + \epsilon_k) > 0$

The original Perceptron reweighing rule (34a) updates the current weight θ_k by adding to it the corresponding entry a_k in the EWC corresponding to the current underlying, winner, and lose form, as stated in (37a). The truncated Perceptron reweighing rule (34b) can therefore be described in EWC notation as in (37b).

(37) a. $\theta_k \leftarrow \theta_k + a_k$ b. $\theta_k \leftarrow \begin{cases} \theta_k + a_k & \text{if } \theta_k + a_k \geq 0 \\ 0 & \text{otherwise} \end{cases}$

These considerations show that the HG error-driven learner (2) can be described compactly in terms of EWCs as in (38). At step (38a), the learner receives the current piece of data in the form of an EWC $\mathbf{a}$, which compactly represents the comparison between the current winner and loser forms. At step (38b), the learner evaluates the performance of the current weight vector $\boldsymbol{\theta} = (\theta_1, \dots, \theta_n)$ on the current piece of data $\mathbf{a} = (a_1, \dots, a_n)$ in terms of one of the two conditions (36), depending on whether we adopt the deterministic or the stochastic implementation. If the condition holds, the algorithm has nothing to learn from the current piece of data $\mathbf{a}$, and just waits for another EWC. If the condition does not hold, then the algorithm updates the current weights at step (38c) according to one of the two reweighing rules (37), depending on whether we adopt the original or the truncated Perceptron implementation. The algorithm then forgets about the current piece of data, loops back to step (38a) and waits for another piece of data.

(38)

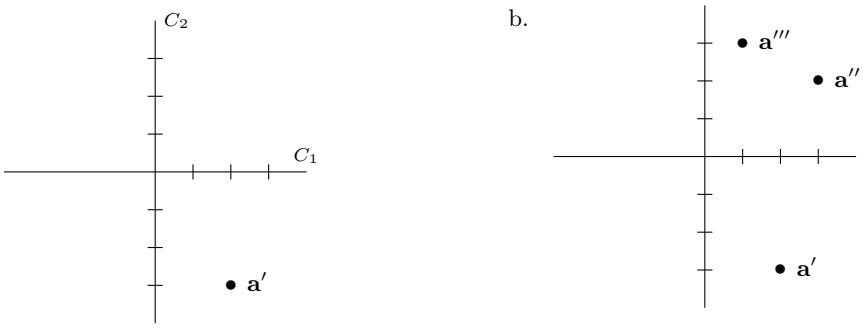


In the rest of this appendix, I provide the general formulation of the theoretical guarantees for error-driven learning for general (not necessarily binary) constraints. The reformulation (38) in EWC will prove particularly convenient.

A.3. **Radius and margin of the training data**

Let $\mathbf{A}$ be the set of training EWCs fed to the learner at step (38b). Error-bounds for the HG error-driven learner are stated in terms of properties of this training set $\mathbf{A}$, namely its *radius* and *margin*. This subsection introduces this notions geometrically; for an analytic definition, see appendix C.1. Consider the case where there are only $n = 2$ constraints C_1 and C_2. A generic EWC $\mathbf{a} = (a_1, a_2)$ thus consists of only two components: 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 plotted in the cartesian plane, by adopting the convention that the horizontal axis corresponds to constraint C_1 and the vertical axis corresponds to constraint C_2. To illustrate, consider the EWC $\mathbf{a} = [2, -3]$ which consists of the constraint differences 2 and -3 corresponding to the two constraints C_1 and C_2, respectively. It can be plotted as the dot in (40a): the horizontal coordinate of the dot is 2, because the horizontal axis corresponds to constraint C_1; the vertical coordinate is -3, because the vertical axis corresponds to constraint C_2.

(39) a. b.

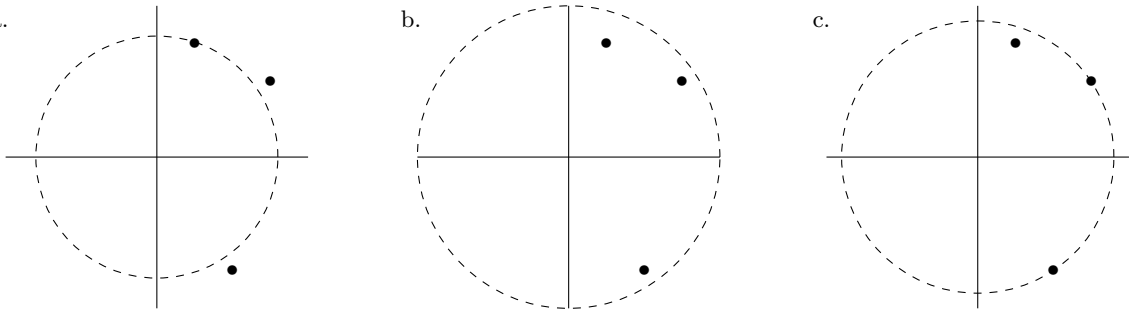


Suppose that the training set $\mathbf{A}$ consists of this EWC $\mathbf{a}'$ plus two more, say $\mathbf{a}'' = [3, 2]$ and $\mathbf{a}''' = [0, 3]$. The latter two EWCs can be plotted analogously, yielding the geometric representation (39b) of the training set $\mathbf{A}$.

Consider now the circumferences centered in the origin, such as those plotted in (40). Depending on the radius of these circumferences, three scenarios may arise. One possibility is that the radius is small, so that the corresponding circumference fails at containing all training EWCs, as illustrated in (40a). Another possibility is that the radius is large, so that the corresponding circumference contains all training EWCs with a good slack, as illustrated in (40b). A final possibility is that the radius coincides with the distance from the origin of the training EWC further away,

so that the corresponding circumference contains all training EWCs without any slack, as illustrated in (40c). The radius of the latter circumference is univocally determined, and is called the radius $\rho(\mathbf{A})$ of the training set $\mathbf{A}$.

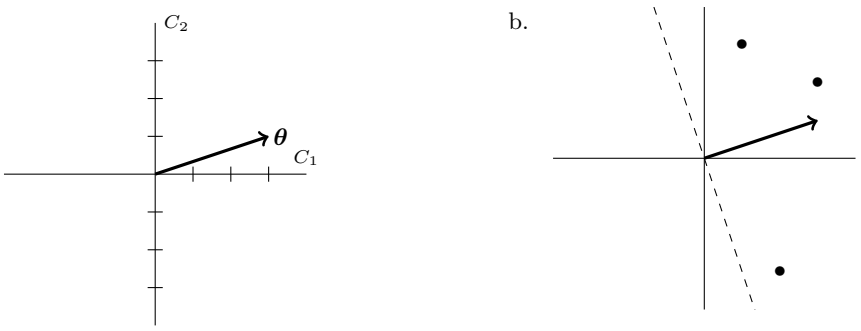
(40) a. b. c.



The extension from the case with only $n = 2$ to an arbitrary number n of constraints is conceptually straightforward. The analytic definition of the radius for an arbitrary number n of constraints is provided in (50a) in appendix C.1.

Since there are only $n = 2$ constraints, a generic ranking vector $\boldsymbol{\theta} = (\theta_1, \theta_2)$ consists of only two components: the ranking values θ_1 and θ_2 of the two constraints C_1 and C_2. A ranking vector can thus be plotted in the cartesian plane as follows. Keeping with the convention that the horizontal and vertical axes correspond to the constraints C_1 and C_2 respectively, we look at the point in the plane whose horizontal coordinate is θ_1 and whose vertical coordinate is θ_2, and we then draw an arrow which starts at the origin and ends on that point. To illustrate, I plot in (41a) the ranking vector $\boldsymbol{\theta} = (3, 1)$, which assign the weight $\theta_1 = 3$ to constraint C_1 and the weight $\theta_2 = 1$ to constraint C_2.

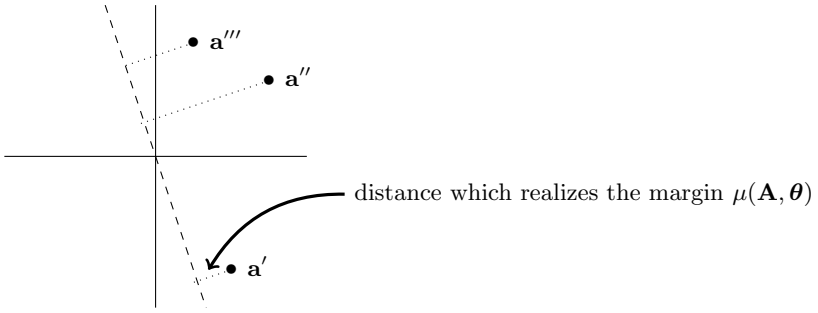
(41) a. b.



The *decision line* corresponding to a ranking vector $\boldsymbol{\theta}$ is the line perpendicular to it which passes through the origin, as illustrated in (41b). This line splits the plane into two regions, one of which contains the weight vector itself. It turns out that the condition (36a) that an EWC is HG consistent with a weight vector can be given the following geometric interpretation: that EWC lies in the half-plane which contains the weight vector itself. To illustrate, the plot in (41b) shows that the weight vector considered is consistent with all three training EWCs, as they all lie in the same half-plane which contains the weight vector itself.

Consider the segment which starts at a training EWC and falls perpendicularly on the decision line individuated by the weight vector, as illustrated in (42) with the dotted segments. The length of each of these dotted segments represents the *distance* of the corresponding EWC from the decision line. As noted above, a weight vector is consistent with an EWC provided the corresponding decision line leaves that EWC on the same side of the plane as the weight vector itself. Intuitively, the distance of an EWC from the decision line can thus be interpreted as a measure of the "degree of consistency" of that EWC with that weight vector. Thus, although the weight vector plotted in (42) is consistent with both EWCs $\mathbf{a}'$ and $\mathbf{a}''$ (because they both lie on the same side of the decision line as the weight vector itself), the former EWC has a smaller degree of consistency than the latter. Indeed, a small perturbation in the weight vector will slightly rotate the decision line and might thus affect consistency with the closer EWC $\mathbf{a}'$ but not consistency with the EWC $\mathbf{a}''$ which sits at a greater distance. Since we are interested in worst-case analyses, we focus on the "dangerous" EWC in the training dataset, namely the EWC which has the smallest distance from the decision line and thus has the smallest degree of consistency. In the case of figure (42), that is the EWC $\mathbf{a}'$. The distance of that EWC from the decision line is called the *margin* of the training set $\mathbf{A}$ with respect to the weight vector $\boldsymbol{\theta}$, and is denoted by $\mu(\mathbf{A}, \boldsymbol{\theta})$. It captures the worst-case degree of consistency achieved by the ranking vector $\boldsymbol{\theta}$ over the training set $\mathbf{A}$.

(42)



Of course, different weight vectors induce different decision lines that in turn differ because of their distances from the training EWCs. For instance, if the decision line in (42) is rotated slightly clockwise (which corresponds to shifting some weight from constraint C_2 to constraint C_1), then the distance of the EWC $\mathbf{a}'$ will increase at the expense of the distance of the EWC $\mathbf{a}'''$. Among all weight vectors consistent with the data set, we thus consider a weight vector $\hat{\boldsymbol{\theta}}$ which achieves the largest distance from the closest EWC, namely whose margin $\mu(\mathbf{A}, \hat{\boldsymbol{\theta}})$ is at least as large as the margin $\mu(\mathbf{A}, \boldsymbol{\theta})$ relative to any other weight vector $\boldsymbol{\theta}$. The margin of any such *optimal* weight vector is called the *margin* of the training set $\mathbf{A}$ and is denoted by $\mu(\mathbf{A})$. As it is clear from this geometric definition, all optimal weight vectors are parallel to each other and thus correspond to the same decision line, which is therefore unique. The margin of the training set captures its intrinsic best degree of consistency. The extension from the case with only $n = 2$ to an arbitrary number n of constraints is conceptually straightforward. The analytic definition of the margin for an arbitrary number n of constraints is provided in (50b) in appendix C.1.

A.4. **Convergence guarantees and error-bounds**

Theorem 1 from section 2 provides convergence guarantees and error-bounds for the deterministic HG error-driven learner with the *original* Perceptron reweighing rule. Here, I recall the more general formulation of the theorem, for general (not necessarily binary) constraints; the proof is recalled for completeness in appendix C.1 (Block 1962; Novikoff 1962; Rosenblatt 1958, 1962; Minsky and Papert 1969; Cristianini and Shawe-Taylor 2000, Theorem 2.3; Cesa-Bianchi and Lugosi 2006, Chp. 12; Mohri et al. 2012, ch. 7). In the case of binary constraints, the square of the radius $\rho(\mathbf{A})$ of the training set $\mathbf{A}$ is upper bounded by the number n of constraints. In this case, the error-bound (43) thus reduces to the error-bound (5) in the formulation of theorem 1 provided in section 2.

Theorem 1 (general version). *Suppose that the HG error-driven learner (38) with the deterministic update condition (36a) and the original Perceptron reweighing rule (37a) is trained on an HG consistent training set* $\mathbf{A}$ *of EWCs. The number of errors made by the learner is at most*

$$\left(\frac{\rho(\mathbf{A})}{\mu(\mathbf{A})}\right)^2 \tag{43}$$

where $\rho(\mathbf{A})$ *and* $\mu(\mathbf{A})$ *are the radius and the margin of the set* $\mathbf{A}$ *of training EWCs.* ■

In order for the error-bound (43) to make sense, the training set $\mathbf{A}$ needs to be contained within a sphere of finite radius $\rho(\mathbf{A})$ (otherwise, the right hand side is infinite and the bound thus trivial). As constraint violations are integers, the EWCs in the training set $\mathbf{A}$ are integer vectors. A sphere of finite radius only contains a finite number of integer vectors. In order for the error-bound (43) to make sense, the training set $\mathbf{A}$ thus needs to be finite.

Theorem 2 from section 2 provides convergence guarantees and error-bounds for the deterministic HG error-driven learner with the *truncated* Perceptron reweighing rule, which guarantees non-negativity of the weights. Here, I provide the more general formulation of the theorem, for general (not necessarily binary) constraints; the proof is provided in appendix C.2. Again, the square of the radius $\rho(\mathbf{A})$ in the numerator of (44) is upper bounded by the number n of constraints in the case of binary constraints, yielding the error-bound (12) in the formulation of theorem 2 provided in section 2.

Theorem 2 (general version). *Suppose that the HG error-driven learner (38) with the deterministic update condition (36a) and the truncated Perceptron reweighing rule (37b) is trained on an HG consistent training set* $\mathbf{A}$ *of EWCs. Let* $\mathbf{E}$ *be the collection of all EWCs whose components are all zeros but for one component which is instead equal to 1. The number of errors made by the learner is at most*

$$\left(\frac{\rho(\mathbf{A})}{\mu(\mathbf{A} \cup \mathbf{E})}\right)^2 \tag{44}$$

where ρ *is the radius of the training set* $\mathbf{A}$ *and* $\mu(\mathbf{A} \cup \mathbf{E})$ *is the margin of the training set* $\mathbf{A}$ *extended with the EWCs in* $\mathbf{E}$*.* ■

Again, in order for the error-bound (44) to make sense, the training set $\mathbf{A}$ needs to be finite. The set $\mathbf{E}$ of additional EWCs plays the role of the dummy data in the informal reasoning sketched in section 2. The margin $\mu(\mathbf{A} \cup \mathbf{E})$ of the training EWCs plus the dummy EWCs can be quite smaller than the margin $\mu(\mathbf{A})$ of the original training set. As a result, the error-bound (44) for the truncated Perceptron reweighing rule can be worse (namely larger)

than the error-bound (43) for the original Perceptron reweighing rule. This is consistent with the fact that the truncated Perceptron performs more errors than the original Perceptron in the simulation results plotted in (21b).

Theorem 4 from section 4 provides convergence guarantees and error-bounds for the *stochastic* HG error-driven learner with ether the original or the truncated Perceptron reweighing rule. Here, I provide the more general formulation of the theorem, for general (not necessarily binary) constraints; the proof is provided in appendix C.3 (Boersma and Pater to appear). In the case of binary constraints, the square of the radius $\rho(\mathbf{A})$ in (45) is upper bounded by the number n of constraints and the constant $\beta(\mathbf{A})$ reduces to 1, yielding the error-bound (27) in the formulation of theorem 4 provided in section 4.

Theorem 4 (general version). *Suppose that the HG error-driven learner (38) with the stochastic update condition (36b) is trained on an HG consistent training set* $\mathbf{A}$ *of EWCs. Suppose that the components of the stochastic vectors* $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)$ *are sampled uniformly in between the values* $-\Delta$ *and* Δ *for some constant* $\Delta \geq 0$*, as described in subsection 4.1. The number of errors made by the learner is at most (45a) and (45b) for the original and the truncated Perceptron reweighing rules (37)*

$$(45)\quad a.\ \left(\frac{\rho(\mathbf{A})}{\mu(\mathbf{A})}\right)^2 + 2n\Delta\frac{\beta(\mathbf{A})}{\mu^2(\mathbf{A})} \qquad b.\ \left(\frac{\rho(\mathbf{A})}{\mu(\mathbf{A}\cup\mathbf{E})}\right)^2 + 2n\Delta\frac{\beta(\mathbf{A})}{\mu^2(\mathbf{A}\cup\mathbf{E})}$$

where $\rho(\mathbf{A})$ *and* $\mu(\mathbf{A})$ *are the radius and the margin of the training dataset* $\mathbf{A}$*,* $\beta(\mathbf{A})$ *is the largest (ignoring sign) component over all EWCs in the training set* $\mathbf{A}$*, and* $\mu(\mathbf{A}\cup\mathbf{E})$ *is the margin of the training set* $\mathbf{A}$ *extended with the set* $\mathbf{E}$ *of EWCs whose components are all zeros but for one which is instead equal to 1.* ■

Appendix B. Simulation details and results

This appendix provides the details of the OT and HG simulations reported in subsections 3.3 and 4.3.

B.1. **Details concerning the counterexample**

In subsection 3.3, I have sketched a counterexample meant to show that the number of errors made by the HG error-driven learner grows astronomically large already for a small number of constraints. In this subsection, I offer a more explicit description of the counterexample. To start, let me recall the case with $n = 5$ constraints. In this case, the unique underlying form x comes with five candidates y and z_1, z_2, z_3, and z_4; the constraint violations are described by the number of stars in (19); and the error-driven learner is trained on the target grammar which maps the underlying form x into the winning candidate y, which therefore beats the four other losing candidates z_1, z_2, z_3, and z_4. This means that the algorithm entertains four possible underlying/winner/loser form triplets, namely (x, y, z_1), (x, y, z_2), (x, y, z_3), and (x, y, z_4). As noted in appendix A.2, these four triplets can be described through the four corresponding EWCs, which are provided in (46a), where empty cells stand for zeros. For instance, the entry corresponding to constraint C_3 and the underlying/winner/loser form triplet (x, y, z_1) is set equal to 1 because that constraint assigns 4 violations to the loser z_1 and 3 violations to the winner y, so that the constraint violation difference is $4 - 3 = 1$. The entry corresponding to constraint C_2 in that same row is set equal to -3 because that constraint assigns 0 violations to the loser z_1 and 3 violations to the winner y, so that the constraint violation difference is $0 - 3 = -3$. And son on.

(46) a.

	C_1	C_2	C_3	C_4	C_5
(x, y, z_1)	1	−3	1	1	
(x, y, z_2)		1	−3	1	1
(x, y, z_3)			1	−2	1
(x, y, z_4)				1	−1

b.

C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}
1	−3	1	1						
	1	−3	1	1					
		1	−3	1	1				
			1	−3	1	1			
				1	−3	1	1		
					1	−3	1	1	
						1	−3	1	1
							1	−2	1
								1	−1

Once we adopt the EWC notation, the extension from the case with $n = 5$ constraints to the case with an arbitrary number n of constraints is straightforward. The EWC matrix for the case with n constraints has $n - 1$ rows; the kth row has a 1 corresponding to the kth constraint, followed by a -3, followed by two more 1's (but for the penultimate and ultimate rows, which have -2 and -1 instead of -3, respectively). To illustrate, I give in (46b) the EWC matrix corresponding to the case with $n = 10$ constraints.

In the OT implementation of error-driven learning, the update condition (13) and the update rule (14) only depend on which constraints prefer the winner and which constraints instead prefer the loser. This crucial information is immediately extracted from the sign of the corresponding constraint violation difference: a constraint is winner-(loser-) preferring provided the corresponding constraint violation difference is positive (negative, respectively). If we mark winner- (loser-) preferring constraints with a W (with an L, respectively), the training data (46) can then be represented as in (47). For instance, constraint C_1 has a W corresponding to the underlying/winner/loser form triplet (x, y, z_1) in (47a) because that constraint is winner-preferring, as the corresponding entry in (46a) is positive. On the contrary, constraint C_2 has an L corresponding to that same row, because it is loser-preferring, as

the corresponding entry in (46a) is negative. These rows of W's and L's are called *elementary ranking conditions* (ERC; Prince 2002).[11]

(47) a.

	C_1	C_2	C_3	C_4	C_5
$(x,\ y,\ \cancel{z_1})$	W	L	W	W	
$(x,\ y,\ \cancel{z_2})$		W	L	W	W
$(x,\ y,\ \cancel{z_3})$			W	L	W
$(x,\ y,\ \cancel{z_4})$				W	L

b.

C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}
W	L	W	W						
	W	L	W	W					
		W	L	W	W				
			W	L	W	W			
				W	L	W	W		
					W	L	W	W	
						W	L	W	W
							W	L	W
								W	L

As illustrated in (47), the ERC matrix for the counterexample corresponding to n constraints has $n-1$ rows; the kth row has a W corresponding to the kth constraint, followed by an L, followed by two more W's (but for the penultimate and ultimate rows, whose L is followed by only one and by no W's, respectively).

B.2. **Details concerning the OT and HG simulations**

The details of the OT simulations reported in subsections 3.3 and 4.3 are as follows: the initial ranking values are all equal to zero; the re-ranking rule (14) is used with the promotion amount p set equal to $p = \frac{1}{2w}$, where w is the number of current winner-preferrers (see below for discussion of this choice of this promotion amount); the algorithm is trained directly on the ERCs, which are sampled uniformly at random. The details of the HG simulations are as follows: the initial weights are all equal to zero; the algorithm is trained directly on the EWCs, which are sampled uniformly at random; the reweighing rules used are the original and the truncated Perceptron reweighing rules defined in (37). For both OT and HG stochastic learners, the components of the stochastic vector $\boldsymbol{\epsilon}$ are sampled uniformly between $-\Delta$ and Δ, with $\Delta = 2$. The OT and HG error-driven learners used for the simulations algorithm are available as the Python scripts `EDRA.py` and `Original/TruncatedPerceptron.py` on the author's webpage, together with the `Excel` files containing the lists of training ERCs and EWCs.

I have considered six choices of the number n of constraints, namely $n = 5, 6, 7, 8, 9, 10$. For each n, I have sampled ten sequences of training data. For each of the ten sequences of data, I have run the OT error-driven ranking algorithm, the HG error-driven learning algorithm with the original Perceptron reweighing rule and the HG error-driven learner with the truncated Perceptron reweighing rule until convergence. I have run both the deterministic and the stochastic implementation of these three learners. Tables 1, 2, and 3 reports the number of errors made before reaching convergence in each of the ten runs considered, for each of the six choices of the number n of constraints, for both the deterministic and the stochastic implementation. The last two columns of these tables provide the mean and the standard deviation of the ten numbers of errors reported in the corresponding row.[12]

The solid line in (20) plots the largest number of errors made by the deterministic implementation of the OT error-driven learner over the ten training sequences, as reported in table 1. For instance, the value plotted by the solid line in (20) corresponding to $n = 8$ is 33, because that is the largest number of errors reported in table 1 corresponding to $n = 8$ constraints and the deterministic implementation of the algorithm. Analogously, the solid (dashed) line in (21b) plots the largest number of errors made by the deterministic implementation of the HG error-driven learner with the original (the truncated, respectively) Perceptron reweighing rule over the ten training sequences, as reported in tables 2 and 3.

The solid line in (32a) plots the difference between the largest number of errors made by the stochastic implementation of the OT error-driven learner over the ten training sequences minus the number of errors made by the deterministic OT error-driven learner in that same run, as computed out of table 1. For instance, the value plotted by the solid line in (32a) for $n = 8$ is $96 = 129 - 33$, because the largest number of errors reported in table 1 for the stochastic OT learner corresponding to $n = 8$ constraints is 129 and the number of errors made in that same run by the deterministic learner is 33. Analogously, the solid (dashed) line in (32b) plots the difference between the largest number of errors made by the stochastic implementation of the HG error-driven learner with the original (the truncated, respectively) Perceptron reweighing rule over the ten training sequences minus the number of errors made by the deterministic learner in that same run, as computed out of tables 2 and 3.

Theorem 3 guarantees efficient convergence of the OT error-driven learner under the assumption that the promotion amount p is *calibrated*, namely it is strictly smaller than the *calibration threshold* ℓ/w, where ℓ is the number of undominated loser-preferring constraints (namely, the number of constraints which are demoted) and w is the number of winner-preferring constraints (namely, the number of constraints which are promoted). This

[11] The ERC matrix in (47a) coincides with the one used in Pater (2008) to construct a counterexample against the GLA's convergence, only with an extra W added on the right in each ERC (but in the bottom two).

[12] The standard deviation of the numbers of errors reported in the last column for the stochastic OT error-driven learner grows with the number n of constraints. For the deterministic implementation, the number of errors is constant over the ten runs corresponding to each choice of the number n of constraints, with the only exception of the case corresponding to $n = 8$ constraints, where we see a slight variation in the number of errors over the ten runs. At the moment, I have no explanation neither for why the number of errors made by the deterministic OT error-driven learner varies in the case corresponding to $n = 8$ constraints nor for why it does not vary in the remaining cases.

means in turn that the promotion amount p can be expressed as in (48), for some constant c in between 0 (included) and 1 (excluded).

$$p = c\frac{\ell}{w} \tag{48}$$

With this choice of the promotion amount, we have the error-bounds (49a) and (49b) for the deterministic and the stochastic OT error-driven learner respectively (Tesar and Smolensky 1998; Magri 2012b, 2013a). These are essentially the bounds (15) and (28) provided by theorems 3 and 5 above, where the quantity $2(1-c)$ has been called the *calibration* of the promotion amount and the quantity $n(n-1)$ has been approximated with n^2 to keep the notation simple.

$$\begin{aligned} &\text{a.} && \text{number of errors made by the deterministic OT learner} && \leq && \frac{1}{2(1-c)}n(n-1) \\ &\text{b.} && \text{number of errors made by the stochastic OT learner} && \leq && \frac{1}{2(1-c)}n(n-1) + \frac{\Delta}{(1-c)}n(n-1) \end{aligned} \tag{49}$$

In the simulations considered here, each training ERC has only one loser-preferring constraint. Thus, the number ℓ of undominated loser-preferring constraints is always equal to $\ell = 1$. The choice of the promotion amount $p = \frac{1}{2w}$ used in the simulations thus corresponds to the general scheme (48) with the constant c set equal to $c = 1/2$. The bound (49a) thus boils down to $n(n-1)$, which is indeed the quantity plotted by the dashed line in (20). Furthermore, the second term of the bound (49b) boils down to $4n(n-1)$, which is indeed the quantity plotted by the dashed line in (32a).

Appendix C. Technical details

C.1. Proof of the convergence theorem 1 for the original Perceptron

Throughout this appendix C, $\langle \cdot, \cdot \rangle$ is the *Euclidean scalar product*, defined by $\langle \mathbf{v}, \mathbf{w} \rangle = \sum_{i=1}^{n} v_i w_i$ for any pair of vectors $\mathbf{v} = (v_1, \dots, v_n)$ and $\mathbf{w} = (w_1, \dots, w_n)$; $\|\cdot\|$ is the *Euclidean norm*, defined by $\|\mathbf{v}\|^2 = \langle \mathbf{v}, \mathbf{v} \rangle = \sum_{i=1}^{n} v_i^2$; they are connected through the *Cauchy-Schwartz inequality* $\langle \mathbf{v}, \mathbf{w} \rangle \leq \|\mathbf{v}\|\|\mathbf{w}\|$. The HG deterministic update condition (36a) between a weight vector $\boldsymbol{\theta}$ and an EWC $\mathbf{a}$ can be expressed in terms of the Euclidean scalar product as $\langle \boldsymbol{\theta}, \mathbf{a} \rangle > 0$. The radius $\rho(\mathbf{A})$ and the margin $\mu(\mathbf{A})$ of a training set $\mathbf{A}$ of EWCs, which were defined geometrically in appendix A.3, can now be expressed analytically for an arbitrary number n of constraints as in (50).

$$\begin{aligned} &\text{a.} && \rho(\mathbf{A}) = \max_{\mathbf{a} \in \mathbf{A}} \|\mathbf{a}\| \\ &\text{b.} && \mu(\mathbf{A}) = \max_{\boldsymbol{\theta} \neq \mathbf{0}} \mu(\boldsymbol{\theta}, \mathbf{A}) \text{ where } \mu(\mathbf{A}, \boldsymbol{\theta}) = \min_{\mathbf{a} \in \mathbf{A}} \frac{\langle \boldsymbol{\theta}, \mathbf{a} \rangle}{\|\boldsymbol{\theta}\|} \end{aligned} \tag{50}$$

In this subsection, I recall the proof of the convergence theorem 1 for the original Perceptron, repeated below.

Theorem 1 (general version). *Suppose that the HG error-driven learner (38) with the deterministic update condition (36a) and the original Perceptron reweighing rule (37a) is trained on an HG consistent training set* $\mathbf{A}$ *of EWCs. The number of errors made by the learner is at most*

$$\left(\frac{\rho(\mathbf{A})}{\mu(\mathbf{A})} \right)^2 \tag{51}$$

where $\rho(\mathbf{A})$ *and* $\mu(\mathbf{A})$ *are the radius and the margin of the training set* $\mathbf{A}$ *of EWCs, as computed in (50).* ■

Proof. As current EWCs that do not trigger an update can be ignored, I assume without loss of generality that an update is performed at each time t. Let $\boldsymbol{\theta}_{t+1}$ be the weight vector obtained at time t by updating the current weight vector $\boldsymbol{\theta}_t$ in response to the current EWC $\mathbf{a}_t$ according to the original Perceptron reweighing rule (37a). The proof has three parts. The *first* part of the proof estimates the norm of the current weight vector entertained by the algorithm. To start, note that the norm of the updated weight vector $\boldsymbol{\theta}_{t+1}$ can be bound as in (52) in terms of the norm of the current weight vector $\boldsymbol{\theta}_t$. In step (52a), I have used the definition (37a) of the original Perceptron reweighing rule. In step (52b), I have used the identity $\|\mathbf{v} + \mathbf{w}\|^2 = \|\mathbf{v}\|^2 + \|\mathbf{w}\|^2 + 2\langle \mathbf{v}, \mathbf{w} \rangle$. In step (52c), I have upper bounded by dropping the term $\langle \boldsymbol{\theta}_t, \mathbf{a}_t \rangle$, which is negative by the hypothesis that the current weight vector $\boldsymbol{\theta}_t$ is not consistent with the current EWC $\mathbf{a}_t$. In step (52d), I have upper bounded the norm of the current EWC $\mathbf{a}_t$ with the radius $\rho = \rho(\mathbf{A})$ over all training EWCs.

$$\begin{aligned} \|\boldsymbol{\theta}_{t+1}\|^2 &\overset{(a)}{=} \|\boldsymbol{\theta}_t + \mathbf{a}_t\|^2 \\ &\overset{(b)}{=} \|\boldsymbol{\theta}_t\|^2 + \|\mathbf{a}_t\|^2 + 2\langle \boldsymbol{\theta}_t, \mathbf{a}_t \rangle \\ &\overset{(c)}{\leq} \|\boldsymbol{\theta}_t\|^2 + \|\mathbf{a}_t\|^2 \\ &\overset{(d)}{\leq} \|\boldsymbol{\theta}_t\|^2 + \rho^2 \end{aligned} \tag{52}$$

$$\|\boldsymbol{\theta}_t\|^2 \leq t\rho^2 \tag{53}$$

Since the inequality (52) holds at every time t and since the initial weight vector has null norm, then we obtain the inequality (53), which concludes the first part of the proof, showing that the squared norm of the current weight vector $\boldsymbol{\theta}_t$ grows with time t at most linearly.

Consider now a weight vector $\boldsymbol{\theta}$ which realizes the margin (50b). The *second* part of the proof estimates the scalar product between the latter weight vector and the current weight vector entertained by the algorithm. To start, note that the scalar product between the weight vector $\boldsymbol{\theta}$ and the updated weight vector $\boldsymbol{\theta}_{t+1}$ can be bound as in (54) in terms of the scalar product between that same weight vector $\boldsymbol{\theta}$ and the current weight vector $\boldsymbol{\theta}_t$. In step (54a), I have used the definition (37a) of the original Perceptron reweighing rule. In step (54b), I have used the linearity of the scalar product. In step (54c), I have lower bounded the quantity $\langle \boldsymbol{\theta}, \mathbf{a}_t \rangle / \|\boldsymbol{\theta}\|$ with the smallest such quantity over all training EWCs, namely the margin $\mu = \mu(\mathbf{A})$ of the training set $\mathbf{A}$.

$$\begin{aligned} \frac{\langle \boldsymbol{\theta}, \boldsymbol{\theta}_{t+1} \rangle}{\|\boldsymbol{\theta}\|} &\overset{(a)}{=} \frac{\langle \boldsymbol{\theta}, \boldsymbol{\theta}_t + \mathbf{a}_t \rangle}{\|\boldsymbol{\theta}\|} \\ &\overset{(b)}{=} \frac{\langle \boldsymbol{\theta}, \boldsymbol{\theta}_t \rangle}{\|\boldsymbol{\theta}\|} + \frac{\langle \boldsymbol{\theta}, \mathbf{a}_t \rangle}{\|\boldsymbol{\theta}\|} \\ &\overset{(c)}{\geq} \frac{\langle \boldsymbol{\theta}, \boldsymbol{\theta}_t \rangle}{\|\boldsymbol{\theta}\|} + \mu \end{aligned} \tag{54}$$

$$\frac{\langle \boldsymbol{\theta}, \boldsymbol{\theta}^t \rangle}{\|\boldsymbol{\theta}\|} \geq t\mu. \tag{55}$$

Since the inequality (54) holds at every time t and since the initial weight vector is null, then we obtain the inequality (55), which concludes the second part of the proof, showing that the scalar product between the weight vector $\boldsymbol{\theta}$ and the current weight vector $\boldsymbol{\theta}_t$ grows with time t at least linearly.

The *third* part of the proof connects the two inequalities (53) and (55) as in (56). In step (56a), I have used (55). In step (56b), I have used the Cauchy-Schwartz inequality. In step (56c), I have used (53).

$$\begin{aligned} \big(t\mu\big)^2 &\overset{(a)}{\leq} \left(\frac{\langle \boldsymbol{\theta}, \boldsymbol{\theta}_t \rangle}{\|\boldsymbol{\theta}\|} \right)^2 \\ &\overset{(b)}{\leq} \frac{\|\boldsymbol{\theta}\|^2 \|\boldsymbol{\theta}_t\|^2}{\|\boldsymbol{\theta}\|^2} \\ &= \|\boldsymbol{\theta}_t\|^2 \\ &\overset{(c)}{\leq} t\rho^2 \end{aligned} \tag{56}$$

The chain of inequalities (56) entails in particular that $t \leq \rho^2/\mu^2$, namely that the number of updates t is finite. □

C.2. **Proof of the convergence theorem 2 for the truncated Perceptron**

The convergence theorems 1 and 2 for the original and truncated Perceptron assume consistency of the set $\mathbf{A}$ of EWCs the algorithms are trained on. This assumption means in turn that there exists a vector of non-negative weights consistent with $\mathbf{A}$. Although none of these weights can be negative, some of them could be null. The following lemma guarantees the existence of a consistent vector of weighs which are all strictly positive (namely neither negative nor null). This lemma will be used below for the proof of the convergence theorem 2 for the truncated Perceptron.

Lemma 1. *A finite set $\mathbf{A}$ of HG-consistent EWCs is in particular HG-consistent with a vector of weights which are all strictly positive.* ■

Proof. The hypothesis that $\mathbf{A}$ is HG-consistent means that there exists a vector $\boldsymbol{\theta} = (\theta_1, \dots, \theta_n)$ of non-negative weights $\theta_k \geq 0$ such that $\langle \boldsymbol{\theta}, \mathbf{a} \rangle > 0$ for every EWC $\mathbf{a} \in \mathbf{A}$. If all the weights happen to be strictly positive (i.e., $\theta_k > 0$), then the claim is proven. Otherwise, assume that some weights are null. To simplify the notation, let Ω be the set of those indices k such that the corresponding weight θ_k is strictly positive and let $\overline{\Omega}$ be its complement, as defined in (57).

$$\text{a. } \Omega = \Big\{ k \in \{1, \dots, n\} \,\Big|\, \theta_k > 0 \Big\} \qquad \text{b. } \overline{\Omega} = \Big\{ k \in \{1, \dots, n\} \,\Big|\, \theta_k = 0 \Big\} \tag{57}$$

I will now construct another weight vector $\overline{\boldsymbol{\theta}} = (\overline{\theta}_1, \dots, \overline{\theta}_n)$ which has all positive weights $\overline{\theta}_k > 0$ and furthermore is HG-consistent with $\mathbf{A}$ as well. Let the constants A and B be defined as in (58), which makes sense because of the assumption that the training set $\mathbf{A}$ of EWCs is finite. The constant A is strictly positive, because the original weight vector $\boldsymbol{\theta}$ is HG-consistent with every EWC $\mathbf{a}$ in $\mathbf{A}$. The constant B is instead strictly negative, because at least one EWC needs to have a negative entry (otherwise the claim is trivial).

$$\text{a. } A = \min_{\mathbf{a} \in \mathbf{A}} \langle \boldsymbol{\theta}, \mathbf{a} \rangle \qquad \text{b. } B = \min_{\mathbf{a} = (a_1, \dots, a_n) \in \mathbf{A}} \min_k a_k \tag{58}$$

Define the new weight vector $\overline{\boldsymbol{\theta}} = (\overline{\theta}_1, \dots, \overline{\theta}_n)$ as in (59). The weights thus defined are all strictly positive as desired, because the constant A is strictly positive and the constant B is strictly negative. In general, A is a small value and $|B|$ is a large value. Thus we have effectively slightly perturbed the original weight vector $\boldsymbol{\theta}$ by replacing its null weights with a small positive value.

$$\overline{\theta}_k = \begin{cases} \theta_k & \text{if } k \in \Omega \\ -\dfrac{A}{2(n-1)B} & \text{if } k \in \overline{\Omega} \end{cases} \tag{59}$$

The scalar product between the perturbed weight vector $\overline{\boldsymbol{\theta}}$ and an arbitrary EWC $\mathbf{a}$ in $\mathbf{A}$ can be computed as in (60), which shows that $\overline{\boldsymbol{\theta}}$ is HG-consistent with an arbitrary EWC $\mathbf{a}$ in $\mathbf{A}$.

$$\begin{aligned} \langle \overline{\boldsymbol{\theta}}, \mathbf{a} \rangle &= \sum_{k \in \Omega} \overline{\theta}_k a_k + \sum_{k \in \overline{\Omega}} \overline{\theta}_k a_k \\ &\overset{(a)}{=} \sum_{k \in \Omega} \theta_k a_k - \sum_{k \in \overline{\Omega}} \frac{A}{2(n-1)B} a_k \\ &\overset{(b)}{=} \sum_{k \in \Omega} \theta_k a_k + \sum_{k \in \overline{\Omega}} \theta_k a_k - \sum_{k \in \overline{\Omega}} \frac{A}{2(n-1)B} a_k \\ &= \langle \boldsymbol{\theta}, \mathbf{a} \rangle - \sum_{k \in \overline{\Omega}} \frac{A}{2(n-1)B} a_k \\ &\overset{(c)}{\geq} A - \sum_{k \in \overline{\Omega}} \frac{A}{2(n-1)B} a_k \\ &\overset{(d)}{\geq} A - \sum_{k \in \overline{\Omega}} \frac{A}{2(n-1)B} B \\ &\geq A - \sum_{k \in \overline{\Omega}} \frac{A}{2(n-1)} \\ &\overset{(e)}{\geq} A - \frac{A}{2} \\ &> 0 \end{aligned} \tag{60}$$

In step (60a), I have used the position (59). In step (60b), I have added the quantity $\sum_{k \in \overline{\Omega}} \theta_k a_k$, which is null because the weights θ_k corresponding to indices $k \in \overline{\Omega}$ are all null. In step (60c), I have lower-bounded by replacing $\langle \boldsymbol{\theta}, \mathbf{a} \rangle$ with the smallest possible value A. In step (60d), I have lower-bounded by replacing a_k with its smallest possible value B (this step is licit, because a_k is multiplied by a positive coefficient, since B is negative). In step (60e), I have used the fact that the original weight vector $\boldsymbol{\theta}$ can contain at most $n-1$ null weights (at least one weight needs to be non-null in order for $\boldsymbol{\theta}$ to yield a strictly positive scalar product with the EWCs in $\mathbf{A}$), so that the sum over $\overline{\Omega}$ has at most $n-1$ terms. □

Using the preceding lemma, I can now straightforwardly formalize the reasoning sketched in subsection 2.3 into a proof of the convergence theorem 2 for the truncated Perceptron, as shown in the rest of this subsection.

Theorem 2 (general version). *Suppose that the HG error-driven learner (38) with the deterministic update condition (36a) and the truncated Perceptron reweighing rule (37b) is trained on an HG consistent training set* $\mathbf{A}$ *of EWCs. Let* $\mathbf{E}$ *be the collection of all EWCs whose components are all zeros but for one component which is equal to 1. The number of errors made by the learner can be bound as follows*

$$\left(\frac{\rho(\mathbf{A})}{\mu(\mathbf{A} \cup \mathbf{E})} \right)^2 \tag{61}$$

where $\rho(\mathbf{A})$ *is the radius of the training set* $\mathbf{A}$ *and* $\mu(\mathbf{A} \cup \mathbf{E})$ *is the margin of the training set* $\mathbf{A}$ *extended with the set* $\mathbf{E}$ *of "dummy" EWCs.* ■

Proof. By reasoning as in subsection 2.3, any run of the HG error-driven learner with the truncated Perceptron reweighing rule on a training set $\mathbf{A}$ of EWCs can be mimicked with a run of the algorithm with the original Perceptron reweighing rule on the extended set of EWCs $\mathbf{A} \cup \mathbf{E}$. In fact, suppose that the truncated Perceptron leaves a weight θ_k at zero while the original Perceptron demotes it down to, say, -5. Then, the original Perceptron can be forced to bring it back to zero by feeding it for five times with the EWC in $\mathbf{E}$ which has all components equal to zero but for the kth component which is equal to 1. In other words, the EWCs in $\mathbf{E}$ play the role of the "dummy data" considered in subsection 2.3. The worst-case number of errors $T_{\text{truncated}}(\mathbf{A})$ made by the truncated Perceptron on the training set $\mathbf{A}$ can thus be bound as in (62) in terms of the number of errors $T_{\text{original}}(\mathbf{A} \cup \mathbf{E})$ made by the original Perceptron on the extended training set $\mathbf{A} \cup \mathbf{E}$.

$$T_{\text{truncated}}(\mathbf{A}) \leq T_{\text{original}}(\mathbf{A} \cup \mathbf{E}) \tag{62}$$

I can assume that the training set $\mathbf{A}$ has a finite radius $\rho(\mathbf{A})$, as the bound (61) would otherwise be infinite and the theorem thus trivial. Furthermore, the EWCs in $\mathbf{A}$ have integer components, because their components represent constraint violation differences. I can thus conclude that the training set $\mathbf{A}$ is finite. As it is furthermore HG consistent by hypothesis, lemma 1 ensures that there exists a vector $\boldsymbol{\theta}$ of strictly positive weights which is HG-consistent with $\mathbf{A}$. Since any vector of strictly positive weights is HG-consistent with the EWCs in $\mathbf{E}$, I conclude that this weight vector $\boldsymbol{\theta}$ is HG-consistent with the extended training set $\mathbf{A} \cup \mathbf{E}$. The Perceptron convergence theorem 1 thus applies, ensuring that the worst-case number of errors $T_{\text{original}}(\mathbf{A} \cup \mathbf{E})$ made by the original Perceptron on the extended data set $\mathbf{A} \cup \mathbf{E}$ can be bound in terms of its radius and margin, as stated in (63).

$$(63)\quad T_{\text{original}}(\mathbf{A}\cup\mathbf{E}) \leq \left(\frac{\rho(\mathbf{A}\cup\mathbf{E})}{\mu(\mathbf{A}\cup\mathbf{E})}\right)^2$$

The radius of the extended training set $\mathbf{A}\cup\mathbf{E}$ is equal to the radius on the original training set $\mathbf{A}$, as computed in (64). In the first equality, I have used the definition (50a) of the radius; in the second equality, I have used the fact that the vectors $\mathbf{e}\in\mathbf{E}$ are unit vectors, namely $\|\mathbf{e}\|=1$. Finally in the third equality, I have used the fact that all the EWC $\mathbf{a}\in\mathbf{A}$ have integer components, so that $\|\mathbf{a}\|\geq 1$.

$$(64)\quad \rho(\mathbf{A}\cup\mathbf{E}) = \max\left\{\max_{\mathbf{a}\in\mathbf{A}}\|\mathbf{a}\|,\ \max_{\mathbf{e}\in\mathbf{E}}\|\mathbf{e}\|\right\} = \max\left\{\max_{\mathbf{a}\in\mathbf{A}}\|\mathbf{a}\|,\ 1\right\} = \max_{\mathbf{a}\in\mathbf{A}}\|\mathbf{a}\| = \rho(\mathbf{A})$$

The claim follows by combining (62), (63), and (64). □

The identity (64) shows that the radius $\rho(\mathbf{A}\cup\mathbf{E})$ of the extended training set $\mathbf{A}\cup\mathbf{E}$ coincides with the radius $\rho(\mathbf{A})$ of the original training set $\mathbf{A}$. This is not true for the margin: the margin $\mu(\mathbf{A}\cup\mathbf{E})$ of the extended training set can be quite smaller than the margin $\mu(\mathbf{A})$ of the original training set, so that the error-bound (61) for the truncated Perceptron can be worse (namely, larger) than the error-bound (51) for the original Perceptron.

C.3. **Proof of the convergence theorem 4 for the (original and truncated) stochastic Perceptron**

This subsection reviews Boersma and Pater's (to appear) proof of the convergence theorem 4 for the HG stochastic error-driven learner with the (original or truncated) Perceptron reweighing rule, repeated below.

Theorem 4. *Suppose that the HG error-driven learner (38) with the stochastic update condition (36b) is trained on an HG consistent training set* $\mathbf{A}$ *of EWCs. Suppose that the components of the stochastic vectors* $\boldsymbol{\epsilon}=(\epsilon_1,\dots,\epsilon_n)$ *are sampled uniformly in between the values* $-\Delta$ *and* Δ *for some constant* $\Delta\geq 0$*, as described in subsection 4.1. The number of errors made by the learner is at most (45a) and (45b) for the original and the truncated Perceptron reweighing rules (37)*

$$(65)\quad a.\ \left(\frac{\rho(\mathbf{A})}{\mu(\mathbf{A})}\right)^2 + 2n\Delta\frac{\beta(\mathbf{A})}{\mu^2(\mathbf{A})} \qquad b.\ \left(\frac{\rho(\mathbf{A})}{\mu(\mathbf{A}\cup\mathbf{E})}\right)^2 + 2n\Delta\frac{\beta(\mathbf{A})}{\mu^2(\mathbf{A}\cup\mathbf{E})}$$

where $\beta=\beta(\mathbf{A})$ *is the largest constraint difference (ignoring sign) over the training set* $\mathbf{A}$*:*

$$(66)\quad \beta=\beta(\mathbf{A}) = \max_{\mathbf{a}=(a_1,\dots,a_n)\in\mathbf{A}}\ \max_k |a_k|$$

and furthermore $\rho(\mathbf{A})$ *and* $\mu(\mathbf{A})$ *are the radius and the margin of the training dataset* $\mathbf{A}$ *and* $\mu(\mathbf{A}\cup\mathbf{E})$ *is the margin of the training set* $\mathbf{A}$ *extended with the set* $\mathbf{E}$ *of EWCs whose components are all zeros but for one which is instead equal to 1.* ■

Proof. To start, consider the case of the original Perceptron reweighing rule (37a). Again, assume without loss of generality that the current ranking vector is always updated and denote by $\boldsymbol{\theta}_t$ the current ranking vector entertained by the algorithm at time t. The chain of inequalities in (54) in the proof of the original Perceptron converge theorem 1 carries over from the deterministic to the stochastic implementation. Thus, the bound (55) also holds for the stochastic implementation, and is therefore repeated in (67).

$$(67)\quad \frac{\langle\boldsymbol{\theta},\boldsymbol{\theta}_t\rangle}{\|\boldsymbol{\theta}\|}\geq t\mu.$$

This bound says that the scalar product between a weight vector $\boldsymbol{\theta}$ consistent with the training EWCs and the current weight vector $\boldsymbol{\theta}_t$ grows faster than t.

The chain of inequalities in (52) in the proof of the original Perceptron converge theorem 1 instead does *not* carry over from the deterministic to the stochastic implementation. It thus needs to be modified as follows. To start, note that the chain of inequalities in (68) holds, where $\boldsymbol{\theta}_t$ is the current weight vector and $\boldsymbol{\epsilon}$ is the current stochastic vector. The first inequality holds because of the assumption that the stochastic consistency condition (36b) has failed at time t triggering an update. The last inequality holds because of the definition (66) of the bound β together with the assumption that the components of the stochastic vector $\boldsymbol{\epsilon}$ are bounded between $-\Delta$ and Δ.

$$(68)\quad 0\geq\langle\boldsymbol{\theta}_t+\boldsymbol{\epsilon},\mathbf{a}\rangle = \langle\boldsymbol{\theta}_t,\mathbf{a}\rangle + \langle\boldsymbol{\epsilon},\mathbf{a}\rangle \geq \langle\boldsymbol{\theta}_t,\mathbf{a}\rangle - n\Delta\beta$$

The chain of inequalities in (52) in the proof of the original Perceptron converge theorem 1 can now be modified as in (69) for the stochastic Perceptron. The only novelty is that in the penultimate step, I have used the inequality $\langle\boldsymbol{\theta}_t,\mathbf{a}\rangle\leq n\Delta\beta$ obtained in (68).

$$(69)\quad \begin{aligned}\|\boldsymbol{\theta}_{t+1}\|^2 &= \|\boldsymbol{\theta}_t+\mathbf{a}_t\|^2\\ &= \|\boldsymbol{\theta}_t\|^2+\|\mathbf{a}_t\|^2+2\langle\boldsymbol{\theta}_t,\mathbf{a}_t\rangle\\ &\leq \|\boldsymbol{\theta}_t\|^2+\|\mathbf{a}_t\|^2+2n\Delta\beta\\ &\leq \|\boldsymbol{\theta}_t\|^2+\rho^2+2n\Delta\beta\end{aligned}$$

Since the inequality (69) holds at every time t and since the initial weight vector has null norm, then we obtain the inequality (70), which shows that the squared norm of the current weight vector $\boldsymbol{\theta}_t$ grows with time t at most linearly.

(70) $\|\boldsymbol{\theta}^t\|^2 \leq t(\rho^2 + 2n\Delta\beta)$

Finally, the error-bound (65a) for the stochastic HG error-driven learner with the original Perceptron reweighing rule is obtained by combining the two inequalities (67) and (70) exactly as done in (56) for the case of the deterministic implementation.

As seen in appendix C.2, the convergence theorem 1 for the deterministic HG learner with the original Perceptron reweighing rule straightforwardly yields the convergence theorem 2 for the deterministic learner with the truncated Perceptron reweighing rule. Nothing changes when we switch from the deterministic to the stochastic implementation: again, convergence of the stochastic HG error-driven learner with the original Perceptron reweighing rule straightforwardly yields convergence for the stochastic learner with the truncated Perceptron reweighing rule. □

C.4. **Computing the margin of the counterexample**

Consider again the counterexample constructed in subsection 3.3 to show that the number of errors made by the HG error-driven learner can grow astronomically large. In Appendix B.1, I have provided a reformulation of this counterexample in EWC notation. Let $\mathbf{A}_n$ be the EWC matrix for the case with n constraints. To illustrate, the EWC matrices $\mathbf{A}_5$ and $\mathbf{A}_{10}$ corresponding to the cases $n = 5$ and $n = 10$ are explicitly provided in (46). Since the constraints used in this counterexample are not binary, then the generalized version of theorem 1 from appendix A.4 is needed in order to bound the number of errors made by the HG error-driven learner on the training dataset $\mathbf{A}_n$. The error-bound (43) provided by this theorem is ρ_n^2/μ_n^2, where: $\rho_n = \rho(\mathbf{A}_n)$ is the radius of the training set $\mathbf{A}_n$ corresponding to n constraints and $\mu_n = \mu(\mathbf{A}_n)$ is its margin, as defined geometrically in appendix A.3 and analytically in (50) in appendix C.1.

The squared radius is equal to $\rho_n^2 = 1 + 3^2 + 1 + 1 = 12$, and is therefore a constant which does not depend on the number n of constraints. Vapnik (1998, Theorem 10.2) ensures that the squared inverse $1/\mu_n^2$ of the margin of the dataset $\mathbf{A}_n$ coincides with the (unique) solution of the optimization problem (71) in the variable $\boldsymbol{\theta} \in \mathbb{R}^n$.

(71) minimize: $\|\boldsymbol{\theta}\|^2$
subject to: $\langle \boldsymbol{\theta}, \mathbf{a} \rangle \geq 1$ for every EWC $\mathbf{a}$ in the training set $\mathbf{A}_n$

As (71) is a convex quadratic 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 (21a).

C.5. **Computing the best-case number of errors on the counterexample**

In this appendix, I assume that a weight vector $\boldsymbol{\theta}$ is a column vector and that an EWC $\mathbf{a}$ is a row vector. A certain number m of EWCs are stacked one on top of the other into a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ with m rows (as many rows as there are EWCs) and n columns (as many columns as there are constraints). Given a matrix $\mathbf{A}$, $\mathbf{A}^{\mathrm{T}}$ stands for its transpose and $\mathbf{A}\boldsymbol{\theta}$ for the row-by-column matrix product. 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 $\mathbf{w}$.

Let again $\mathbf{A}_n$ be the EWC matrix which describes the counterexample constructed in subsection 3.3 corresponding to n constraints. The final weight vector $\boldsymbol{\theta}$ entertained at convergence by the Perceptron algorithm trained on the EWC set $\mathbf{A}_n$ must be HG-consistent with each training EWC, namely must satisfy the strict matrix inequality (72a), where $\mathbf{0}$ is a column vector of $n - 1$ entries all equal to 0.

(72) a. $\mathbf{A}_n\boldsymbol{\theta} > \mathbf{0}$
b. $\boldsymbol{\theta} = \mathbf{A}_n^{\mathrm{T}}\boldsymbol{\alpha}$
c. $\mathbf{A}_n\mathbf{A}_n^{\mathrm{T}}\boldsymbol{\alpha} > \mathbf{0}$
d. $\mathbf{A}_n\mathbf{A}_n^{\mathrm{T}}\boldsymbol{\alpha} \geq \mathbf{1}$

The algorithm starts from the null weight vector and at each update adds to the current weight vector the row of the matrix $\mathbf{A}_n$ corresponding to the EWC which is triggering the current update. This means in turn that the final weight vector is the sum of the rows of the matrix $\mathbf{A}$, 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 (72b), 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 $\mathbf{A}_n$. This vector $\boldsymbol{\alpha}$ is also called the *dual* vector corresponding to the *primal* weight vector $\boldsymbol{\theta}$. By combining (72a) and (72b), I obtain the strict matrix inequality (72c), which involves the dual not the primal vector. Finally, since $\boldsymbol{\alpha}$ is an integer vector and $\mathbf{A}_n$ is an integer matrix, the strict inequality (72c) is equivalent to the inequality (72d), where $\mathbf{1}$ is a column vector of $n - 1$ entries all equal to 1.

The total number of updates performed by the Perceptron algorithm in the run considered 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 (72d). In the end, the number of updates performed by the Perceptron algorithm to reach convergence cannot be smaller than the solution of the optimization problem (73) 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 algorithm.

$$\begin{array}{lll} (73) & \text{minimize:} & \alpha_1 + \ldots + \alpha_{n-1} \\ & \text{subject to:} & \mathbf{A}_n \mathbf{A}_n^{\mathrm{T}} \boldsymbol{\alpha} \geq \mathbf{1} \\ & & \boldsymbol{\alpha} \geq \mathbf{0} \end{array}$$

As (73) 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 with the dotted line in (21b).

References

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

Bane, Max, and Jason Riggle. 2010. Evaluating Strict Domination: The typological consequences of weighted constraints. In *Annual meeting of the Chicago Linguistic Ssociety (CLS) 45*.

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

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

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.

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.

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

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.

Kivinen, Jyrki, Manfred K. Warmuth, and P. Auer. 1997. The Perceptron algorithm versus Winnow: linear versus logarithmic mistake bounds when few input variables are relevant. *Artificial Intelligence* 97:325–343.

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.

Littlestone, N. 1988. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning* 2.4:285–318.

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.

n	**implement.**	**number of errors made in the 10 runs**										**mean**	**SD**
5	deterministic	11	11	11	11	11	11	11	11	11	11	11.0	0.0
	stochastic	20	23	26	11	31	23	27	17	20	17	21.5	5.48
6	deterministic	18	18	18	18	18	18	18	18	18	18	18.0	0.0
	stochastic	36	32	43	23	31	58	30	28	31	54	36.6	10.90
7	deterministic	24	24	24	24	24	24	24	24	24	24	24.0	0.0
	stochastic	65	51	66	74	87	54	32	54	54	75	61.2	14.73
8	deterministic	31	33	31	33	31	33	31	33	31	31	31.8	0.98
	stochastic	92	129	97	109	46	91	78	110	111	44	90.73	26.4
9	deterministic	38	38	38	38	38	38	38	38	38	38	38.0	0.0
	stochastic	161	133	182	153	160	107	126	141	133	192	148.8	24.67
10	deterministic	47	47	47	47	47	47	47	47	47	47	47.0	0.0
	stochastic	173	231	217	221	196	211	196	154	186	230	201.5	24.00

TABLE 1. Number of errors made by the OT error-driven ranking algorithm before convergence

Magri, Giorgio. 2013a. Convergence of error-driven ranking algorithms: extensions and refinements. Manuscript.

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

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

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* 39.2:334–345.

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.

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

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

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

Schölkopf, Bernhard, and Alexander Smola. 2002. *Learning with kernels*. 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, and Paul Smolensky. 1998. Learnability in Optimality Theory. *Linguistic Inquiry* 29:229–268.

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.

n	implement.	number of errors made in the 10 runs										mean	SD
5	deterministic	443	443	443	443	443	443	443	443	443	443	443.0	0.0
	stochastic	594	526	609	526	443	609	609	526	511	526	547.9	52.54
6	deterministic	3391	3391	3339	3160	3339	2966	3197	3160	3160	3160	3226.30	128.78
	stochastic	4636	4688	3890	4158	4337	4531	4315	4315	4084	4248	4320.2	234.99
7	deterministic	21243	21961	22372	21243	21153	22372	21871	21744	21280	21871	21711.0	438.61
	stochastic	27292	28130	27988	26500	28758	27091	24062	27756	25782	30209	27356.8	1593.7
8	deterministic	133428	139533	132236	136045	138288	136045	136698	138506	133210	139751	136374.0	2563.72
	stochastic	168308	176686	174555	163193	177339	174015	163193	176813	163822	160231	169815.5	6408.14
9	deterministic	839247	816109	810922	823813	828782	828782	831608	841855	824031	834368	827951.67	9162.77
	stochastic	1049133	1070129	1054629	1027922	1006364	1038695	1004155	1059817	1033636	1009753	1,035,423.3	22153.18
10	deterministic	5023263	4926701	4979108	5000812	4890064	4908069	4955077	4938344	4938871	4878729	4943903.8	44365.97
	stochastic	6302072	6180108	6178011	6220226	6266090	6109914	6218211	6181172	5934271	6173002	6176307.70	95186.16

TABLE 2. Number of errors made by the HG original Perceptron before convergence

n	implement.	number of errors made in the 10 runs										mean	SD
5	deterministic	617	617	608	576	607	609	604	580	618	611	604.75	14.1
	stochastic	693	936	874	706	630	657	801	619	727	744	738.7	98.61
6	deterministic	4382	4476	4510	4345	4662	4563	4598	4632	4507	4332	4500.7	111.41
	stochastic	5640	5332	5618	5779	5520	5345	5350	5942	5399	5225	5515.0	215.74
7	deterministic	29561	29055	30012	29535	30308	29934	30387	30338	29438	30151	29871.90	428.94
	stochastic	37480	37218	35646	33407	37663	35757	36901	34523	34752	36153	35950.0	1337.79
8	deterministic	185421	186248	184253	182034	186159	185202	186264	184613	180310	180844	184134.8	2144.86
	stochastic	231715	217371	223001	219277	212590	224250	222747	225288	222889	209114	220824.20	6182.39
9	deterministic	1086527	1103322	1102998	1100572	1096704	1096580	1107024	1099137	1103871	1103721	1100045.601	5507.48
	stochastic	1313655	1319071	1383520	1399764	1416778	1397433	1356751	1376207	1397803	1361640	1372262.2	32838.61
10	deterministic	6451243	6494936	6450208	6478230	6507866	6370174	6449933	6361722	6439061	6522585	6452595.8	50648.1
	stochastic	8066792	7735720	7937825	7977033	7926763	7750502	7843547	8050801	8003092	7865444	7915751.9	109775.66

TABLE 3. Number of errors made by the HG truncated Perceptron before convergence