How to keep the HG weights non-negative: the truncated Perceptron reweighing rule*

Giorgio Magri
SFL (CNRS and University of Paris 8) and UiL-OTS (Utrecht University)

ABSTRACT

The literature on error-driven learning in Harmonic Grammar (HG) has adopted the *Perceptron* reweighing rule. Yet, this rule is not suited to HG, as it fails at ensuring non-negative weights. A variant is thus considered which truncates the updates at zero, keeping the weights non-negative. Convergence guarantees and error bounds for the original Perceptron are shown to extend to its truncated variant.

Keywords: Harmonic Grammar, error-driven learning, Perceptron, convergence

1 INTRODUCTION

Language learning is the process of selecting a grammar from a given typology of grammars based on some linguistic data. Assume that the learner maintains a current grammar representing its current hypothesis on the target adult grammar it is being trained on. Training data come in a stream. Whenever the current grammar makes an error on the current piece of training data, it is updated to a slightly different one. The current piece of data is then discarded and the learner waits for the next piece of training data to evaluate the performance of the updated grammar. This learning scheme is called *error-driven* because the learning dynamics is driven by the errors made on the incoming stream of data. This scheme has been thoroughly investigated in the machine learning literature (where it is commonly called *online learning*; for a review, see Kivinen 2003; Cesa-Bianchi and Lugosi 2006,

*I would like to thank Paul Boersma and Nazarré Merchant for useful comments. The research reported in this paper has been supported by a Marie Curie Intra European Fellowship (Grant agreement number: PIEF-GA-2011-301938).

chapters 11, 12; and Mohri *et al.* 2012, ch. 7). Within the language acquisition literature, this learning scheme has been endorsed at least since Wexler and Culicover (1980) for two reasons. First, an error-driven learner describes a sequence of grammars in typological space and thus provides a tool to model child acquisition paths. Second, an error-driven learner does not keep track of previously seen data (the current piece of data is discarded at the end of each iteration) and can thus be used to model the early stages of language acquisition prior to the development of the native language lexicon (such as the early acquisition of phonotactics; Hayes 2004).

The most basic question of the computational theory of error-driven learning concerns *convergence*: is it possible to guarantee that the learner only makes a finite number of errors, so that it describes a finite sequence of grammars in typological space? Convergence is crucial because it means that the learner eventually settles on a final grammar which will never be further updated and thus counts as the grammar *learned* by the algorithm. This paper focuses on convergence of error-driven learning within the framework of *Harmonic Grammar* (HG; Legendre, Miyata, and Smolensky 1998b,a; Smolensky and Legendre 2006).

Within the HG framework, the typology of grammars is parameterized through an assignment of *weights* to a given, finite set of *constraints* which extract the relevant properties of the linguistic data. Whenever the HG error-driven learner makes an error, the constraints are slightly reweighed. The recent HG computational literature has adopted the *Perceptron* (or *delta*) reweighing rule (Pater 2008; 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. This reweighing rule comes with convergence guarantees, reviewed in Section 2: the number of errors is always finite and can be bounded in terms of certain "geometric" properties of the training data (Rosenblatt 1958, 1962; Block 1962; Novikoff 1962; Minsky and Papert 1969; Cesa-Bianchi and Lugosi 2006, chapters 11, 12; Mohri *et al.* 2012, ch. 7).

Despite current practice, the Perceptron is *not* suited to HG. Crucially, HG requires the weights to be non-negative, in order to avoid undesired typological predictions. Yet, the Perceptron does not en-

force non-negativity of the final weights, since the current weights are decreased (as well as increased) throughout learning. A simple solution to this problem is a variant of the Perceptron reweighing rule which truncates the updates at zero, thus ensuring non-negativity of the final weights (Boersma and Pater to appear; Boersma and van Leussen 2014). Although a run of the original and the truncated Perceptron can differ substantially, Section 3 shows that a run of the truncated Perceptron can be "mimicked" with a run of the original Perceptron on a slightly different sequence of data. Convergence guarantees thus extend from the original to the truncated Perceptron. This observation yields the first convergence guarantee for an HG error-driven learner consistent with HG's restriction to non-negative weights. This result is constraint-independent, namely it follows from the HG mode of constraint interaction and thus holds for any constraint set. Section 4 extends the reasoning to the stochastic implementation of HG error-driven learning and to the noisy learning setting.

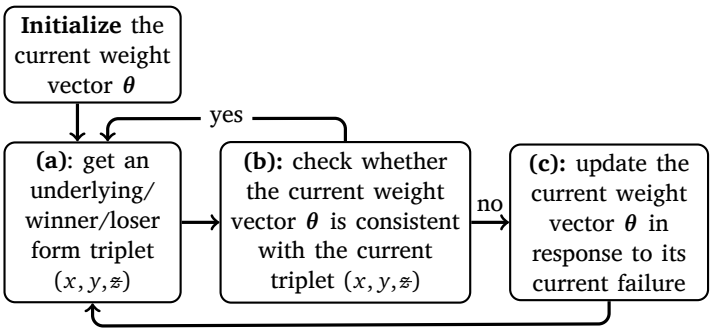
2 THE *ORIGINAL* PERCEPTRON HG LEARNER

This Section reviews the implementation of error-driven learning used in the current HG literature, based on the *Perceptron* reweighing rule.

2.1 *Algorithmic core*

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

(1)



The algorithm maintains a current vector $\boldsymbol{\theta}$ of constraint weights. The analyses reported in this paper are independent of how these weights are initialized; for concreteness, I assume throughout that they are all initialized to zero (the extension to arbitrary initial weights is straightforward). The current weights are then updated by looping through the three steps (1a)–(1c), described in detail below.

2.2 *Data provided at step (1a)*

At step (1a), the learner receives a piece of data. At a minimum, this piece of data consists of a surface form y, say some form which is licit according to the phonotactics corresponding to the target grammar the learner is being trained on. In some applications, the corresponding underlying form x can be assumed to be provided as well. In other applications, the learner needs to be endowed with an additional subroutine to reconstruct the underlying form (e.g., set the current underlying form x identical to the current surface form y; Prince and Tesar 2004; Hayes 2004; Magri 2015). Yet, the analyses reported in this paper are independent of the subroutine for the choice of the underlying form, which I thus assume to be provided in some arbitrary way along with the surface form at step (1a). The mapping (x, y) of the underlying form x to the surface form y must beat the mapping $(x, \not{z})$ of x to any other loser candidate $\not{z}$ (loser candidates are stricken out as a mnemonic) according to the target grammar the learner is being trained on. The learner needs to focus on one such loser candidate $\not{z}$. Usually, this loser candidate is chosen through a proper subroutine.[1] Yet, the analyses reported in this paper are independent of the subroutine for the choice of the loser form, which I thus assume to be provided in some arbitrary way at step (1a) as well. In the end, I assume that the learner is fed at step (1a) a piece of data which consists of an underlying/winner/loser form triplet $(x, y, \not{z})$. The collection of these triplets is called the *training set* (each triplet from the training set can of course be fed multiple times to the learner).

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

2.3 *Consistency condition checked at step (1b)*

At step (1b), the learner checks whether the current weight vector $\boldsymbol{\theta}$ is *consistent* with the current underlying/winner/loser form triplet $(x, y, \mathscr{z})$, namely whether the HG grammar corresponding to the current weight vector $\boldsymbol{\theta}$ manages to make the intended winner y beat the intended loser $\mathscr{z}$ for the underlying form x. To start, assume that all constraints are *binary*, namely they assign at most one violation. In this case, the condition that the intended winner y beats the intended loser $\mathscr{z}$ boils down to the condition (2). This condition says that the sum of the current weights $\theta_1, \theta_2, \ldots$ of the *winner-preferring* constraints collected into the set W (namely those constraints which assign fewer violations to the winner y than to the loser $\mathscr{z}$) is larger than the sum of the weights of the *loser-preferring* constraints collected into the set L (namely those constraints which assign fewer violations to the loser $\mathscr{z}$ than to the winner y).

(2) $$\sum_{h \in W} \theta_h > \sum_{k \in L} \theta_k$$

In the general case of arbitrary (possibly non-binary) constraints, the consistency condition generalizes to (3). The *violation difference* of constraint C_k is the difference between the number of violations $C_k(x, \mathscr{z})$ it assigns to the loser $\mathscr{z}$ minus the number of violations $C_k(x, y)$ it assigns to the winner y (see appendix A.1 for discussion). Condition (3) thus requires the average of the constraint violation differences weighted by the current weights θ_k to be strictly positive.

(3) $$\sum_{k=1}^{n} \Big(\underbrace{C_k(x, \mathscr{z}) - C_k(x, y)}_{\substack{\text{violation difference} \\ \text{of constraint } C_k}} \Big) \theta_k > 0$$

In the case of binary constraints, the consistency condition (3) indeed reduces to (2).

2.4 *Update performed at step (1c)*

If the consistency condition (2)/(3) is satisfied, the current weight vector already predicts that the current winner y beats the current loser $\mathscr{z}$. The learner thus has nothing to learn from the current comparison and loops back to step (1a). Otherwise, the current weight vector $\boldsymbol{\theta}$ needs to be updated at step (1c) in response to its current failure. To start, assume that the constraints are all binary. Failure of

condition (2) suggests that the weights corresponding to the winner-preferring (loser-preferring) constraints are too small (too large). One reasonable update strategy is thus (4), which promotes (demotes) the winner-preferring (the loser-preferring) constraints by a small amount, say 1 for concreteness; for an illustration, see (10) 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.

In the general case of arbitrary (possibly non-binary) constraints, the update rule (4) is generalized as in (5) (Jesney and Tessier 2011; Coetzee and Pater 2008, 2011; Coetzee and Kawahara 2013; Boersma and Pater to appear, among many others). If a constraint C_k is winner-preferring (loser-preferring), its violation difference $C_k(x,z)-C_k(x,y)$ is positive (negative) and its weight is therefore increased (decreased) by the update rule (5). In the case of binary constraints, the update rule (5) indeed reduces to (4).

(5) Update each current weight θ_k by adding the corresponding violation difference $C_k(x,z)-C_k(x,y)$.

After the update, the learner loops back to step (1a), waits for another piece of data, and starts all over again.

2.5 *Convergence*

Boersma and Pater (to appear) note that the HG reweighing rule (5) can be interpreted as the *Perceptron* (or *delta*) update rule. They thus reinterpret the convergence guarantees for the Perceptron (Rosenblatt 1958, 1962; Block 1962; Novikoff 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) as the following Theorem 1 on convergence of the HG error-driven learner.[2] See Magri (to appear) for discussion of the error bound (6).

[2] Boersma and Pater (to appear) call the learner just described the (deterministic) *HG-GLA*. I prefer the name *HG (Perceptron) error-driven learner*, thus keeping the acronym "GLA" for a specific implementation of OT error-driven learning, characterized by the fact that the promotion amount is set equal to the demotion amount.

Theorem 1 *The HG error-driven learner (1) with the HG update condition (3) and the Perceptron reweighing rule (5) converges: the number of errors is bounded by*

$$\left(\frac{\textit{radius of the training data}}{\textit{margin of the training data}}\right)^2 \tag{6}$$

when the training set consists of underlying/winner/loser form triplets which are all consistent with some HG grammar and have bounded violation differences.

Theorem 1 provides an error bound (6) which depends on some "geometric" properties of the training data, namely their *radius* and their *margin (of separability)*. Here is the idea in a nutshell. The training set consists of underlying/winner/loser form triplets $(x, y, \cancel{z})$. For each of these training triplets, collect into a vector the n violation differences assigned by the phonological constraints $C_1, \ldots, C_n$ to that triplet. The resulting vector can be thought of as a point in the cartesian n-dimensional space. The *radius* of the training data which appears in the numerator of (6) is the radius of the smallest sphere which contains all the vectors of constraint violation differences that the learner is trained on, as explained in Appendix A.2. Of course, the error bound (6) only makes sense provided that the radius in the numerator is finite, or equivalently that the violation differences are bounded, as indeed required by the statement of the Theorem. That is in particular the case if the number of violations assigned by the constraints is upper bounded by some constant.

The precise definition of the *margin (of separability)* which appears in the denominator of (6) is somewhat involved and is therefore relegated to Appendix A.3. The following intuitive illustration suffices for the rest of the paper. Theorem 1 assumes that the training set is consistent with some HG grammar. Yet, consistent training sets can differ in their *degree of consistency*. The training set has a *high* degree of consistency if it is consistent with a certain HG grammar and remains consistent when the corresponding weight vector is tampered with. The training set has instead a *small* degree of consistency if even a slight modification of any consistent weight vector affects consistency. The margin which appears in the denominator of (6) can be interpreted as the degree of consistency of the training set. Intuitively, a training set with a large degree of consistency should be easy to learn:

it should be easy to shoot at a consistent weight vector. A training set with a small degree of consistency should instead be hard to learn: a careful aim is required to shoot precisely at a consistent weight vector. The error bound (6) formalizes this intuition: a training set with a high (low) degree of consistency has a large (small) margin, yielding a small (large) error bound (6) which provides guarantees for better (worse) performance of the HG learner.

3 THE *TRUNCATED* PERCEPTRON HG LEARNER

This section explains why the problem of convergence for HG error-driven learning is still open in the literature and provides a simple and principled solution.

3.1 *The problem of non-negative weights*

Constraints in HG are always interpreted as expressing penalties, never rewards. Hence, constraint weights need to be enforced to satisfy the non-negativity condition (7) in order for HG to avoid undesired typological predictions, whereby less marked structures are mapped to more marked ones (Legendre *et al.* 2006; Keller 2000).

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

Here is an elementary counterexample which illustrates the importance of this non-negativity condition. Suppose that the constraint set contains the markedness constraint NoVoice against voiced obstruents and the identity faithfulness constraint Ident[voice] for voicing. Suppose furthermore that the underlying form /ta/ comes with the two surface candidates [ta] and [da]. If the two constraints are allowed to take on negative weights (say $\theta_{\text{NoVoice}} = -3$ and $\theta_{\text{Ident}} = -1$), the corresponding HG grammar maps the voiceless stop to a voiced one, whereby an unfaithful mapping yields no gain in markedness.

Despite the non-negativity condition (7) being a crucial component of HG's mode of constraint interaction, the Perceptron update rule (4)/(5) used in the current literature does not in any way guarantee that the current and final weights entertained by the algorithm satisfy this non-negativity condition (7). 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 initial weights equal to zero, namely to start right at 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 might foul the learner into learning phonotactically unrestrictive final weights. In conclusion, the Perceptron reweighing rule (4)/(5) does not yield a proper HG error-driven learner. The rest of this section develops a solution to this problem.[3]

3.2 *The truncated Perceptron reweighing rule*

To start, assume for concreteness that all constraints are binary. A natural strategy to enforce non-negativity of the current and final weights is to switch from the *original* Perceptron update rule (4) to the *truncated* Perceptron update rule (8). 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 (8) leaves that weight unchanged; for an illustration, see (12) below.

[3] A different solution to this problem is to use the *Winnow* algorithm 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. Unfortunately, convergence guarantees for Winnow only hold when the amount of reweighing (also called the *plasticity* or the *step size*) has been properly chosen in a way that crucially depends on the margin of the training data. Since the margin is not known beforehand, 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 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 (it only differs because the current weights are not normalized, contrary to what is 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 Winnow's convergence proof), they report that the number of errors is significantly smaller than with the Perceptron on their test cases. 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 training sets.

(8) 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.*

In the general case of arbitrary (possibly non-binary) constraints, the truncated Perceptron reweighing rule becomes (9). This is the original update rule (5) apart from the additional "unless" clause in italics, meant to prevent the weights from ever turning negative.[4] In the case of binary constraints, (9) indeed reduces to (8).

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

Boersma and Pater (to appear, p. 19) and Boersma and van Leussen (2014, section 5) report encouraging simulation results with this truncated reweighing rule. But what about its theoretical guarantees? Theorem 1 guarantees that the learner with the *original* Perceptron update rule (4)/(5) can only make a finite number of errors and furthermore provides an explicit error bound. What about the truncated Perceptron update rule (8)/(9)? The two update rules are superficially similar; yet, they describe quite different algorithms. Indeed, suppose that the current weights are all initialized to zero. The original Perceptron update rule will then perform lots of demotions below zero that the truncated Perceptron is forbidden to mimic. As a result, the sequence of grammars entertained by the original Perceptron will turn out to be quite different from the sequence of grammars entertained by the truncated Perceptron. Is there any way to extend the theoretical guarantees that hold for the original Perceptron to its truncated variant?

3.3 *Sketch of the analysis on a concrete example*

Consider three constraints IDENTVOICEONSET (which requires preservation of voicing in onset position), IDENTVOICE (which requires

[4] A slight variant of (8)/(9) is as follows: when updating a weight 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.

preservation of voicing in an arbitrary position), and NoVoice (which militates against obstruent voicing). Assume that the underlying form /da/ only comes with the two candidates [da] and [ta]. Suppose that at a certain iteration of the HG learner, the current weight of the constraint IdentVoiceOnset is equal to zero, thus barely satisfying the non-negativity condition (7). Suppose for concreteness that the current weights of the other two constraints IdentVoice and NoVoice are instead positive, say equal to 7 and 3 respectively, as indicated by the weight vector on the left hand side of (10).

(10) *Update by the original Perceptron:*

$$\begin{array}{r} \textsc{IdentVoiceOnset} \\ \textsc{IdentVoice} \\ \textsc{NoVoice} \end{array} \begin{bmatrix} 0 \\ 7 \\ 3 \end{bmatrix} \xrightarrow{(\text{/da/, [ta], } \cancel{\text{[da]}})} \begin{bmatrix} -1 \\ 6 \\ 4 \end{bmatrix}$$

Suppose that the learner is trained on a target grammar which bans voiced obstruents across the board. The learner is thus fed the underlying form /da/ together with the corresponding intended winner [ta] and the faithful loser ~~[da]~~ , as indicated by the label on top of the arrow in (10).

The markedness constraint NoVoice prefers the winner candidate [ta] while the two faithfulness constraints IdentVoiceOnset and IdentVoice prefer the loser candidate ~~[da]~~, as shown in (11).

(11)

Input: /da/	IdentVoiceOnset $\theta = 0$	IdentVoice $\theta = 7$	NoVoice $\theta = 3$
a. [ta]	$*$(L)	$*$(L)	
b. ☞ ~~[da]~~			$*$(W)

The weight $\theta_{\textsc{NoVoice}} = 3$ of the winner-preferring constraint is not larger than the sum $\theta_{\textsc{IdentVoiceOnset}} + \theta_{\textsc{IdentVoice}} = 0 + 7$ of the weights of the two loser-preferring faithfulness constraints. Condition (2) therefore fails and the current weights need to be updated. The original Perceptron update rule (4) prescribes that the weights of the two loser-preferring constraints IdentVoiceOnset and IdentVoice each be decreased by 1 while the weight of the winner-preferring constraint NoVoice be increased by 1, obtaining the updated weight vector on the right hand side of (10). The weight of the loser-preferring constraint IdentVoiceOnset has thus

dropped to the negative value −1, in violation of the non-negativity condition (7). If this were the final update, the learner would have effectively failed.

The update according to the truncated Perceptron update rule (9) in this same scenario is described in (12). The weight of the winner-preferring constraint NoVoice is increased by 1 and the weight of the loser-preferring constraint IdentVoice is decreased by 1, just as in the case of the original Perceptron. The crucial difference is that the weight of the constraint IdentVoiceOnset is left unchanged in order to prevent it from turning negative, despite the fact that the constraint is loser-preferring.

(12) *Update by the truncated Perceptron:*

$$\begin{array}{r} \textsc{IdentVoiceOnset} \\ \textsc{IdentVoice} \\ \textsc{NoVoice} \end{array} \begin{bmatrix} 0 \\ 7 \\ 3 \end{bmatrix} \xrightarrow{(/\text{da}/,\ [\text{ta}],\ \cancel{[\text{da}]})} \begin{bmatrix} 0 \\ 6 \\ 4 \end{bmatrix}$$

Crucially, the update (12) by the truncated Perceptron can be analyzed as the sequence (13) of two updates by the original Perceptron. At the first update (13a), the original Perceptron is fed the piece of data (/da/, [ta], ~~[da]~~) as in (10). Thus, in particular, the weight of the loser-preferring constraint IdentVoiceOnset is demoted to −1.

(13) *Sequence of two updates by the original Perceptron:*

$$\begin{array}{r} \textsc{IdentVoiceOnset} \\ \textsc{IdentVoice} \\ \textsc{NoVoice} \end{array} \underbrace{\begin{bmatrix} 0 \\ 7 \\ 3 \end{bmatrix} \xrightarrow{(/\text{da}/,\ [\text{ta}],\ \cancel{[\text{da}]})} \begin{bmatrix} -1 \\ 6 \\ 4 \end{bmatrix}}_{\text{a.}} \underbrace{\xrightarrow{(x,\ y,\ \cancel{z})} \begin{bmatrix} 0 \\ 6 \\ 4 \end{bmatrix}}_{\text{b.}}$$

Immediately afterwards, the original Perceptron is fed with the "dummy" piece of data described in (14). This piece of data consists of the underlying form x together with the corresponding winner candidate y and the loser candidate $\cancel{z}$. The only constraint which distinguishes between these two candidates is IdentVoiceOnset, which prefers the winner. There are no loser-preferring constraints. In other words, the violation differences corresponding to this triplet $(x, y, \cancel{z})$ are all null apart from the one corresponding to the constraint IdentVoiceOnset which is equal to +1. Condition (2) fails: the right hand side is null (because there are no loser-preferring constraints)

and the left-hand side is negative (because IDENTVOICEONSET is the only winner-preferring constraint, and its current weight -1 is negative). The original Perceptron thus performs the update (13b): the weight of the winner-preferring constraint IDENTVOICEONSET is increased by 1 back to zero, and no other weights are modified.

(14)

Input: x	IDENTVOICEONSET $\theta = -1$	IDENTVOICE $\theta = 6$	NOVOICE $\theta = 4$
a. y			
b. ☞ $\cancel{z}$	$*$(W)		

In the end, the final weights after the two updates (13) by the original Perceptron are identical to the final weights after the single update (12) by the truncated Perceptron.

3.4 *Convergence of the truncated Perceptron reweighing rule*

The analysis of this specific example extends to the general case. Any update according to the truncated Perceptron reweighing rule (9) can be analyzed as a sequence of updates according to the original Perceptron reweighing rule (5), namely the update triggered by the actual piece of data followed by some updates triggered by *dummy* data which undo the illicit demotions that yielded negative weights. These dummy data have a winner-preferring constraint but no loser-preferring constraints. In other words, their constraint violation differences are all null apart for one, which is equal to $+1$. If the training data are consistent with some HG grammar, the training plus the dummy data are consistent as well (see Appendix A.5). Of course, these dummy data have no phonological meaning. Indeed, I am *not* suggesting that the set of phonological forms should be extended with these dummy data. These artificial data only play a role in the analysis (not in the simulations) of the truncated Perceptron.

Let me take stock. The convergence Theorem 1 for the original Perceptron ensures convergence whenever the training data are consistent. A run of the truncated Perceptron can be analyzed as a run of the original Perceptron on the training data extended with dummy data which undo forbidden reweighing. Furthermore, consistency of the original training data guarantees consistency of the extended data. The convergence Theorem 1 for the original Perceptron thus yields the

analogous convergence Theorem 2 for the truncated Perceptron. Appendix A.6 formalizes the reasoning sketched above into a proof. See Magri (to appear) for more discussion of the error bound (15).

Theorem 2 *Let the set of dummy data consist of underlying/winner/loser form triplets whose violation differences are all equal to zero apart from one which is equal to* $+1$. *The HG error-driven learner (1) with the HG update condition (3) and the* truncated *Perceptron reweighing rule (9) converges: the number of errors is bounded by*

$$\left(\frac{\textit{radius of the training data}}{\textit{margin of the training plus dummy data}}\right)^2 \tag{15}$$

when the training set consists of underlying/winner/loser form triplets which are all consistent with some HG grammar and have bounded violation differences.

The error bound (15) for the truncated Perceptron only differs from the error bound (6) for the original Perceptron because the latter has the margin of only the training data at the denominator while the former has the margin of the training plus dummy data. Let me comment on this difference. The margin of a training set quantifies its degree of consistency. Intuitively, extending a training set with additional data can only shrink the degree of consistency (any grammar consistent with the extended training set is also consistent with the original one, but not vice versa; see Appendix A.3). Hence, the margin of the original training set extended with the dummy data which appears in the error bound (15) for the truncated Perceptron is equal to or smaller than the margin of just the original training set which appears in the error bound (6) for the original Perceptron. The error bound (15) for the truncated Perceptron is therefore worse than (namely, at least as large as) the error bound (6) for the original Perceptron. The difference between the two margins quantifies the price that needs to be paid for HG's assumption (7) of non-negative weights.

4 EXTENSION TO THE STOCHASTIC IMPLEMENTATION AND THE NOISY SETTING

This section extends the analysis of the truncated Perceptron to the stochastic implementation and the noisy learning setting.

4.1 Stochastic implementation

The implementation of error-driven learning considered so far is called *deterministic*, to distinguish it from the *stochastic* implementation (Boersma 1997, 1998; Boersma and Hayes 2001; Coetzee and Pater 2008, 2011; Coetzee and Kawahara 2013; Boersma and Pater to appear; Jarosz 2013). Intuitively, the latter differs because the current piece of data is compared not with the current grammar but with a variant thereof sampled from a neighborhood of the current grammar. This intuition can be formalized as follows. At step (1b), the deterministic HG error-driven learner checks whether the current weights $\theta_1, \ldots, \theta_n$ satisfy the update condition (2) or (3), depending on whether the constraints are binary or possibly gradient. The only innovation of the stochastic implementation is that this update condition is checked not for the current weights $\theta_1, \ldots, \theta_n$ but for the *stochastic weights* $\theta_1 + \epsilon_1, \ldots, \theta_n + \epsilon_n$, obtained by adding to the current weights certain values $\epsilon_1, \ldots, \epsilon_n$ sampled independently from each other according to the same underlying distribution. In other words, the learner checks the *stochastic update conditions* (16) or (17), depending on whether the constraints are binary or possibly gradient.

(16) $$\sum_{h \in W} (\theta_h + \epsilon_h) > \sum_{k \in L} (\theta_k + \epsilon_k)$$

(17) $$\sum_{k=1}^{n} \Big(\underbrace{C_k(x, \mathscr{z}) - C_k(x, y)}_{\text{violation difference}} \Big) (\theta_k + \epsilon_k) > 0$$

These stochastic values ϵ_k are usually sampled according to a gaussian distribution with zero mean and small variance (Boersma 1997, 1998; Boersma and Hayes 2001). Since the tails of the gaussian distribution decrease exponentially fast, these stochastic values are bounded *with high probability* between some thresholds $-\Delta$ and $+\Delta$. From an analytical perspective, it is nonetheless convenient to assume they are *deterministically* bounded, namely sampled according to a distribution concentrated between $-\Delta$ and $+\Delta$. The analyses carry over with high probability to the gaussian distribution. The algorithm (1) with the update condition (16)/(17) at step (1b) is called the HG

stochastic error-driven learner[5] (Boersma 1997, 1998; Boersma and Hayes 2001; Coetzee and Pater 2008, 2011; Coetzee and Kawahara 2013; Boersma and Pater to appear, Jarosz 2013).

For simplicity, assume that all constraints are binary (the reasoning extends to the general case). The stochastic update condition (16) can be rewritten as in (18), where the value ϵ on the right hand side is the sum of those stochastic values $\epsilon_1, \epsilon_2, \ldots$ which correspond to the loser-preferring constraints minus the sum of those stochastic values which instead correspond to the winner-preferring constraints.[6]

(18) $$\sum_{h \in W} \theta_h - \sum_{k \in L} \theta_k > \epsilon$$

The stochastic update condition (18) is thus almost identical to the deterministic update condition (2), repeated in (19) with all the terms rearranged on the left. The only difference is that zero on the right hand side of (19) is replaced by ϵ in (18). Yet, ϵ cannot be much different from zero, since it is the sum of numbers sampled between $-\Delta$ and $+\Delta$.

(19) $$\sum_{h \in W} \theta_h - \sum_{k \in L} \theta_k > 0$$

Since the deterministic and stochastic implementations only differ for the update conditions and since these conditions differ only minimally, the convergence Theorem 1 for the original deterministic Perceptron extends to the stochastic variant. Based on this reasoning, Boersma and Pater (to appear) obtain the convergence guarantees for the stochastic *original* Perceptron summarized in Theorem 3. The error bound (20) is the sum of two terms. The first term (20a) coincides with the error bound (6) for the deterministic HG learner. The second term (20b) thus quantifies the number of additional errors due to the stochastic implementation.

Theorem 3 *Assume that the stochastic values $\epsilon_1, \ldots, \epsilon_n$ of the n constraints are sampled independently in between $-\Delta$ and $+\Delta$ for some con-*

[5] It is called instead the *Noisy HG-GLA* in Boersma and Pater (to appear). As explained in footnote 2, I prefer not to use the acronym "GLA" in the context of HG. Furthermore, I prefer "stochastic" over "noisy", in order to avoid any confusion between the stochastic implementation considered here and the noisy learning setting considered in Subsection 4.2.

[6] Namely: $\epsilon = \sum_{k \in L} \epsilon_k - \sum_{h \in W} \epsilon_h$.

stant $\Delta \geq 0$. The HG error-driven learner with the stochastic update condition and the original Perceptron reweighing rule converges: the number of errors is bounded by

$$\text{(20)}\quad \underbrace{\left(\frac{\textit{radius of training data}}{\textit{margin of training data}}\right)^2}_{\text{(a)}} + \underbrace{2n\Delta\frac{\textit{largest absolute value of violation differences}}{\left(\textit{margin of training data}\right)^2}}_{\text{(b)}}$$

when the training set consists of underlying/winner/loser form triplets which are all consistent with some HG grammar and have bounded violation differences.

By reasoning as in Subsection 3.3, this result extends to the stochastic *truncated* Perceptron, yielding the following Theorem 4. Again, the only difference between the two error bounds (20) and (21) for the original and the truncated Perceptron is that the denominator of the former has the margin of the training data while the denominator of the latter has the margin of the training plus dummy data.

Theorem 4 *Let the set of dummy data consist of underlying/winner/loser form triplets whose violation differences are all equal to zero apart from one which is equal to $+1$. Assume that the stochastic values $\epsilon_1, \ldots, \epsilon_n$ of the n constraints are sampled in between $-\Delta$ and $+\Delta$ for some constant $\Delta \geq 0$. The HG error-driven learner (1) with the stochastic update condition and the* truncated *Perceptron reweighing rule converges: the number of errors is bounded by*

$$\text{(21)}\quad \left(\frac{\textit{radius of training data}}{\begin{array}{c}\textit{margin of training}\\ \textit{plus dummy data}\end{array}}\right)^2 + 2n\Delta\frac{\textit{largest absolute value of violation differences}}{\left(\begin{array}{c}\textit{margin of training}\\ \textit{plus dummy data}\end{array}\right)^2}$$

when the training set consists of underlying/winner/loser form triplets which are all consistent with some HG grammar and have bounded violation differences.

4.2 *Noisy learning setting*

A realistic learning setting needs to allow for the possibility that the (possibly infinite) sequence of *pristine* training data generated by some target grammar has been interspersed with data *corrupted* by transmission noise or production errors (Gibson and Wexler 1994, p. 410;

Frank and Kapur 1996, p. 625; Boersma and Hayes 2001, pp. 66–67; Bíró 2006, among many others). No assumptions are made on the corrupted data, apart from there being only a finite number of them.[7] The classical error bound for the Perceptron algorithm in this noisy learning setting is due to Freund and Schapire (1999) (building on Klasner and Simon 1995; see also Mohri *et al.* 2012, ch. 7 for a textbook treatment). The shape of their bound is recalled in (22). The precise definition of the quantity which appears as the second term in the numerator is somewhat involved and therefore relegated to Appendix A.7. What is crucial is that this quantity is null when there are no corrupted training data and grows with the number of corrupted data. The error bound (22) differs from the error bound (6) for the noise-free setting because of this additional quantity, which thus quantifies the additional number of errors due to the corrupted training data. Subsequent improvements of Freund and Shapire's error bound (Shalev-Shwartz and Singer 2005; Mohri and Rostamizadeh 2013) do not alter its basic shape (22).

Theorem 5 *Consider the HG error-driven learner with the deterministic update condition*[8] *and the original Perceptron reweighing rule. Suppose it is trained on a (possibly infinite) sequence of* pristine *training data consisting of underlying/winner/loser form triplets which are all consistent with some HG grammar and have bounded violation differences. Suppose that this sequence is interspersed with a finite number of arbitrary* corrupted *data. The number of errors made by the learner on this corrupted training sequence is bounded by:*

$$(22)\quad \left(\frac{\begin{array}{c}\textit{radius of the pristine}\\ \textit{plus corrupted data}\end{array} + \begin{array}{c}\textit{a quantity which depends}\\ \textit{on the corrupted data}\end{array}}{\textit{margin of the pristine data}} \right)^2$$

[7] Indeed, if an infinite number of corrupted data were allowed, the worst case number of errors would always be infinite: whenever the learner rests on a current hypothesis, we can prompt it to perform yet another update by maliciously crafting an appropriate piece of corrupted data.

[8] It is only for simplicity that the analysis of the noisy learning setting is limited to the deterministic implementation. Theorems 3 and 5 can be easily combined, yielding an error bound for the HG stochastic learner in the noisy setting.

By reasoning as in Subsection 3.3, this result extends to the truncated Perceptron, yielding the following Theorem 6. Again, the only difference between the two error bounds (22) and (23) for the original and the truncated Perceptron is that the denominator of the former has the margin of the pristine training data while the denominator of the latter has the margin of the pristine training data plus the dummy data.

Theorem 6 *Let the set of dummy data consist of underlying/winner/loser form triplets whose violation differences are all equal to zero apart from one which is equal to* $+1$. *Consider the HG error-driven learner with the deterministic update condition and the* truncated *Perceptron reweighing rule. Suppose it is trained on a (possibly infinite) sequence of* pristine *training data consisting of underlying/winner/loser form triplets which are all consistent with some HG grammar and have bounded violation differences. Suppose that this sequence is interspersed with a finite number of arbitrary* corrupted *data. The number of errors made by the learner on this training sequence can be bounded by:*

$$\left(\frac{\begin{array}{c}\textit{radius of the pristine}\\ \textit{plus corrupted data}\end{array} + \begin{array}{c}\textit{a quantity which depends}\\ \textit{on the corrupted data}\end{array}}{\textit{margin of the pristine plus dummy data}} \right)^2 \tag{23}$$

5 CONCLUSIONS

The current HG error-driven learning literature has adopted the Perceptron reweighing rule. Yet, this reweighing rule is not suited to HG, as it does not guarantee non-negativity of the weights. I have thus considered a variant whereby the updates are "truncated" at zero, enforcing non-negativity of the weights in a principled way. A run of the truncated Perceptron can be analyzed as a run of the original Perceptron on the same training sequence interspersed with dummy data used to "undo" the truncated updates. Convergence guarantees for the original Perceptron (Theorem 1), its stochastic implementation (Theorem 3), and its noise robustness (Theorem 5) thus extend to the truncated variant (Theorems 2, 4, and 6). This observation provides the first constraint-independent convergence guarantees for an HG error-driven learner consistent with HG's restriction to non-negative weights.

A APPENDICES

A.1 *Representing the training data as EWCs*

At each iteration, the error-driven learner (1) processes a piece of data which consists of a certain winner candidate y and a certain loser candidate z for a certain underlying form x. Denote by a_k the difference between the number $C_k(x,z)$ of violations assigned by constraint C_k to the loser mapping minus the number $C_k(x,y)$ of violations assigned to the winner-mapping, namely $a_k = C_k(x,z) - C_k(x,y)$. Collect these violation differences corresponding to the constraints $C_1,\ldots,C_n$ into a vector $\mathbf{a} = (a_1,\ldots,a_n)$, called an *elementary weighting condition* (EWC), in analogy with Prince's 2002 *elementary ranking conditions* in Optimality Theory. The consistency condition (3) between a weight vector $\boldsymbol{\theta} = (\theta_1,\ldots,\theta_n)$ and an underlying/winner/loser form triplet is only stated in terms of the violation differences, not in terms of the actual numbers of constraint violations. It can thus be restated as in (24a) in terms of the EWC $\mathbf{a} = (a_1,\ldots,a_n)$ corresponding to the data triplet. Also the original and the truncated Perceptron reweighing rules (5) and (9) are only stated in terms of violation differences, and can thus be restated as in (24b) and (24c) in terms of EWCs.

(24) a. $\displaystyle\sum_{k=1}^{n} a_k\theta_k > 0$

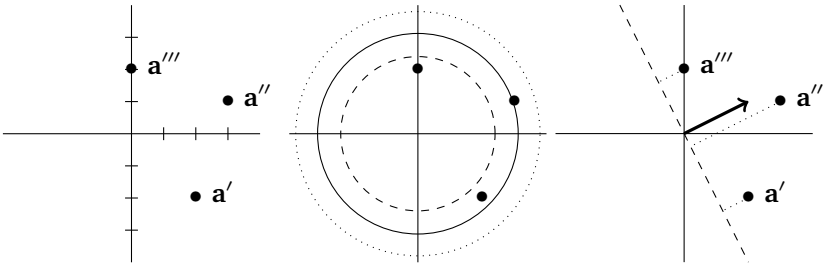
b. $\theta_k \leftarrow \theta_k + a_k$

c. $\theta_k \leftarrow \begin{cases} \theta_k + a_k & \text{if } \theta_k + a_k \geq 0 \\ \theta_k & \text{otherwise} \end{cases}$

In conclusion, the piece of training data (x,y,z) fed to the learner at step (1a) can be represented as an EWC. Throughout this appendix, I thus assume that the HG learner is trained on a sequence of EWCs sampled from a certain *training set* $\mathbf{A}$ of EWCs.

A.2 *Geometric definition of the radius*

Suppose there are only $n = 2$ constraints C_1 and C_2. A generic EWC thus has the shape $\mathbf{a} = (a_1,a_2)$, where a_1 and a_2 are the violation differences corresponding to the two constraints C_1 and C_2, respectively. The EWC can thus be represented with a point in the cartesian plane, through the convention that the horizontal axis corresponds to constraint C_1 and the vertical axis corresponds to constraint C_2. To illustrate, the



a. Example of EWCs b. Example of radius c. Example of margin

Figure 1: Geometric illustration of EWCs (left), radius (center) and margin (right).

EWC set $\mathbf{A} = \{\mathbf{a}', \mathbf{a}'', \mathbf{a}'''\}$ consisting of the three EWCs $\mathbf{a}' = (2, -2)$, $\mathbf{a}'' = (3, 1)$, and $\mathbf{a}''' = (0, 2)$ can be represented as in Figure 1a.

Consider now various circles of different radiuses centered in the origin. The radius could be too small, so that the corresponding circle fails at containing all EWCs in $\mathbf{A}$, as in the case of the dashed circle in Figure 1b. Or the radius could be too large, so that the corresponding circle contains all EWCs with some slack, as in the case of the dotted circle. Or the radius could coincide with the distance from the origin of the EWC furthest away, so that the corresponding circle contains all EWCs without any slack, as in the case of the solid circle. The radius of the latter solid circle in Figure 1b is univocally determined. It is called the *radius of the EWC set* $\mathbf{A}$ and denoted by $\rho(\mathbf{A})$. The extension from $n = 2$ 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 (25a) in Appendix A.4.

A.3 *Geometric definition of the margin*

With only $n = 2$ constraints, a generic weight vector has the shape $\boldsymbol{\theta} = (\theta_1, \theta_2)$: it consists of the weights θ_1 and θ_2 of the two constraints C_1 and C_2. The corresponding *decision line* is the line through the origin which is perpendicular to the arrow which starts at the origin and ends at the point whose horizontal and vertical coordinates are θ_1 and θ_2 respectively. To illustrate, the decision line corresponding to the weight vector $\boldsymbol{\theta} = (2, 1)$ is represented by the dashed line in Figure 1c. The decision line splits the plane into two half planes, one of which contains the arrow. The consistency condition (24a) between a weight vector and an EWC says that the EWC lies in the half-plane which contains the arrow which represents the weight vector. To illustrate,

Figure 1c shows that the weight vector considered is consistent with the EWC set $\mathbf{A} = \{\mathbf{a}', \mathbf{a}'', \mathbf{a}'''\}$, because all three EWCs lie in the half-plane containing the arrow.

The *distance* of an EWC $\mathbf{a}$ from the decision line is the length of the segment which starts at $\mathbf{a}$ and falls perpendicularly on the decision line, represented by the dotted segments in 1c. This distance can be interpreted as the "degree of consistency" of the EWC with (the decision line corresponding to) the weight vector. Thus, although the weight vector plotted in Figure 1c is consistent with both EWCs $\mathbf{a}'$ and $\mathbf{a}''$, the former EWC is closer to the decision line and thus has a smaller degree of consistency than the latter. Indeed, a small perturbation of the weights slightly rotates the decision line and might affect consistency with the closer $\mathbf{a}'$ but not with $\mathbf{a}''$. Since we are interested in worst-case analyses, we focus on the most "dangerous" EWC in the EWC set $\mathbf{A}$, namely the one which is closest to the decision line and thus has the smallest degree of consistency. The distance of that EWC from the decision line is called the *margin* of the EWC set $\mathbf{A}$ with respect to the weight vector $\boldsymbol{\theta}$, and is denoted by $\mu(\mathbf{A}, \boldsymbol{\theta})$. To illustrate, the margin of the EWC set $\mathbf{A} = \{\mathbf{a}', \mathbf{a}'', \mathbf{a}'''\}$ relative to the decision line represented by the dashed line in Figure 1c is the distance of either EWCs $\mathbf{a}'$ or $\mathbf{a}'''$.

Different weight vectors induce different decision lines that in turn differ because of their distances from the various EWCs. Among all weight vectors consistent with the EWC set, consider a weight vector $\widehat{\boldsymbol{\theta}}$ whose decision line achieves the largest distance from the closest EWC, namely whose margin $\mu(\mathbf{A}, \widehat{\boldsymbol{\theta}})$ 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 EWC set $\mathbf{A}$ and is denoted by $\mu(\mathbf{A})$. As is clear from this geometric definition, all optimal weight vectors correspond to the same decision line, which is therefore unique. The extension from $n = 2$ to an arbitrary number n of constraints is conceptually straightforward. The analytic definition of the margin for an arbitrary number n of constraints is provided in (25b) in Appendix A.4.

A.4 *Analytical expression of the radius and the margin*

Let $\langle \cdot, \cdot \rangle$ be 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 $\mathbf{v} = (v_1, \ldots, v_n)$ and $\mathbf{w} = (w_1, \ldots, w_n)$. Let $\| \cdot \|$ be the *Euclidean norm*, defined by $\|\mathbf{v}\|^2 = \langle \mathbf{v}, \mathbf{v} \rangle = \sum_{i=1}^{n} v_i^2$. The consis-

tency condition (24a) between a weight vector $\boldsymbol{\theta}$ and an EWC $\mathbf{a}$ can thus be rewritten as the condition $\langle \boldsymbol{\theta}, \mathbf{a} \rangle > 0$. The radius $\rho(\mathbf{A})$ and the margin $\mu(\mathbf{A})$ of a finite EWC set $\mathbf{A}$, which were defined geometrically in Appendices A.2 and A.3, can now be expressed analytically for an arbitrary number n of constraints as in (25).

(25) a. $\rho(\mathbf{A}) = \max_{\mathbf{a} \in \mathbf{A}} \|\mathbf{a}\|$

b. $\mu(\mathbf{A}) = \max_{\boldsymbol{\theta} \neq \mathbf{0}} \mu(\boldsymbol{\theta}, \mathbf{A})$ where $\mu(\mathbf{A}, \boldsymbol{\theta}) = \min_{\mathbf{a} \in \mathbf{A}} \dfrac{\langle \boldsymbol{\theta}, \mathbf{a} \rangle}{\|\boldsymbol{\theta}\|}$

The assumption that the set $\mathbf{A}$ is finite ensures that the maxima over $\mathbf{A}$ are well defined. This assumption is not restrictive. In fact, all the theorems considered in the paper assume that the training set consists of underlying/winner/loser form triplets with bounded violation differences. Since the violation differences are integers, this boundedness assumption is equivalent to the assumption that the EWC set $\mathbf{A}$ corresponding to the training set is finite.

A.5 *Consistency of the training plus dummy data*

The analysis of the truncated Perceptron sketched in Section 3 relies on the notion of *dummy data*. These are underlying/winner/loser form triplets which have a unique non-zero constraint violation difference, which is equal to +1. The set of EWCs corresponding to these dummy data will be denoted by $\mathbf{E}$. Thus, an EWC $\mathbf{e}$ in $\mathbf{E}$ is a vector which has a unique non-zero component, which is equal to +1.

Denote by $\mathbf{A}$ the set of EWCs corresponding to the underlying/winner/loser form triplets the learner is trained on. Suppose that this training set is consistent with the HG grammar corresponding to some weight vector $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$. Can I conclude that the set $\mathbf{A} \cup \mathbf{E}$ obtained by extending the training set $\mathbf{A}$ with the dummy data $\mathbf{E}$ is consistent with $\boldsymbol{\theta}$ as well? Since each dummy EWC $\mathbf{e}$ has no negative components and the weight vector $\boldsymbol{\theta}$ has nonnegative components, that is indeed the case as long as all the weights θ_k are all different from zero, namely not only non-negative but actually strictly positive. If that is not the case, then consistency with the dummy EWC set $\mathbf{E}$ might fail. For instance, the dummy EWC $\mathbf{e} = (1, 0, \ldots, 0)$ (whose unique non-null component corresponds to constraint C_1) is not consistent with a weight vector $\boldsymbol{\theta}$ which assigns to constraint C_1 a null weight $\theta_1 = 0$ (because $\langle \boldsymbol{\theta}, \mathbf{e} \rangle = 0 \not> 0$). Yet, the following lemma guar-

antees that a consistent EWC set $\mathbf{A}$ is always consistent with weights which are strictly positive (namely neither negative nor equal to zero), as weights which are equal to zero can be slightly increased without compromising consistency. 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 EWCs consistent with some HG grammar is in particular consistent with an HG grammar corresponding to weights which are all strictly positive.*

Proof. The hypothesis that $\mathbf{A}$ is consistent means that there exists a weight vector $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$ 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. Thus, assume that some weights are equal to zero. 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 (26).

(26) $\Omega = \left\{ k \in \{1, \ldots, n\} \,\middle|\, \theta_k > 0 \right\} \qquad \overline{\Omega} = \left\{ k \in \{1, \ldots, n\} \,\middle|\, \theta_k = 0 \right\}$

I will now construct another weight vector $\widehat{\boldsymbol{\theta}} = (\widehat{\theta}_1, \ldots, \widehat{\theta}_n)$ which has all positive weights $\widehat{\theta}_k > 0$ and furthermore is consistent with $\mathbf{A}$ as well. Let the constants A and B be defined as in (27), which makes sense because of the assumption that the training EWC set $\mathbf{A}$ is finite. The constant A is strictly positive, because the original weight vector $\boldsymbol{\theta}$ is 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).

(27) a. $A = \min_{\mathbf{a} \in \mathbf{A}} \langle \boldsymbol{\theta}, \mathbf{a} \rangle$ b. $B = \min_{\mathbf{a} = (a_1, \ldots, a_n) \in \mathbf{A}} \min_k a_k$

Define the new weight vector $\widehat{\boldsymbol{\theta}} = (\widehat{\theta}_1, \ldots, \widehat{\theta}_n)$ as in (28). 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 only slightly perturbed the original weight vector $\boldsymbol{\theta}$ by replacing its null weights with a small positive value.

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

The scalar product between the perturbed weight vector $\widehat{\boldsymbol{\theta}}$ and an arbitrary EWC $\mathbf{a}$ in $\mathbf{A}$ can be computed as in (29), which shows that $\widehat{\boldsymbol{\theta}}$ is consistent with $\mathbf{a}$.

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

In step (29a), I have used the position (28). In step (29b), 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 (29c), I have lower-bounded by replacing $\langle \boldsymbol{\theta}, \mathbf{a} \rangle$ with the smallest possible value A. In step (29d), 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 (29e), 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. ■

A.6 *Proof of the convergence Theorem 2 for the truncated Perceptron*

Using the preceding lemma, I can now straightforwardly formalize the reasoning sketched in Subsection 3.2 into a proof of the convergence Theorem 2 for the truncated Perceptron, restated below in terms of EWCs.

Theorem 2. *Let* $\mathbf{E}$ *be the set of the* dummy *EWCs, whose components are all zeros but for one component which is instead equal to* $+1$*. The HG error-driven learner with the deterministic update condition (24a) and the* truncated *Perceptron reweighing rule (24c) converges: when trained on a finite EWC set* $\mathbf{A}$ *consistent with some HG grammar, the number of errors is bounded by*

$$\left(\frac{\rho(\mathbf{A})}{\mu(\mathbf{A}\cup\mathbf{E})}\right)^2 \tag{30}$$

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 dummy set* $\mathbf{E}$*.*

Proof. By reasoning as in Subsection 3.2, any run of the HG error-driven learner with the truncated Perceptron reweighing rule on a training EWC set $\mathbf{A}$ can be mimicked with a run of the algorithm with the original Perceptron reweighing rule on the extended EWC set $\mathbf{A}\cup\mathbf{E}$. 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 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 3.2. 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 bounded as in (31) 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 $\mathbf{A}\cup\mathbf{E}$.

$$T_{\text{truncated}}(\mathbf{A}) \leq T_{\text{original}}(\mathbf{A}\cup\mathbf{E}) \tag{31}$$

Since the training set $\mathbf{A}$ is finite and consistent with some HG grammar, lemma 1 ensures that it is in particular consistent with a weight vector $\boldsymbol{\theta}$ of strictly positive weights. Since any vector of strictly positive weights is consistent with the EWCs in $\mathbf{E}$, I conclude that this weight vector $\boldsymbol{\theta}$ is 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 EWC set $\mathbf{A}\cup\mathbf{E}$ can be bounded in terms of its radius and margin as in (32).

$$T_{\text{original}}(\mathbf{A}\cup\mathbf{E}) \leq \left(\frac{\rho(\mathbf{A}\cup\mathbf{E})}{\mu(\mathbf{A}\cup\mathbf{E})}\right)^2 \tag{32}$$

The radius of the extended training set $\mathbf{A} \cup \mathbf{E}$ is equal to the radius of the original training set $\mathbf{A}$, as computed in (33). In the first equality, I have used the definition (25a) 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 each EWC $\mathbf{a} \in \mathbf{A}$ has integer components, so that $\|\mathbf{a}\| \geq 1$.

$$\begin{aligned} (33) \quad \rho(\mathbf{A} \cup \mathbf{E}) &= \max\left\{\max_{\mathbf{a}\in\mathbf{A}}\|\mathbf{a}\|,\ \max_{\mathbf{e}\in\mathbf{E}}\|\mathbf{e}\|\right\} = \max\left\{\max_{\mathbf{a}\in\mathbf{A}}\|\mathbf{a}\|,\ 1\right\} \\ &= \max_{\mathbf{a}\in\mathbf{A}}\|\mathbf{a}\| = \rho(\mathbf{A}) \end{aligned}$$

The claim follows by combining (31), (32), and (33). ■

The identity (33) shows that the radius $\rho(\mathbf{A} \cup \mathbf{E})$ of the extended EWC set $\mathbf{A} \cup \mathbf{E}$ coincides with the radius $\rho(\mathbf{A})$ of the original EWC set $\mathbf{A}$. This is not true for the margin: the margin $\mu(\mathbf{A} \cup \mathbf{E})$ of the extended EWC set can be smaller than the margin $\mu(\mathbf{A})$ of the original EWC set.

A.7 *Error-bound for the noisy learning setting*

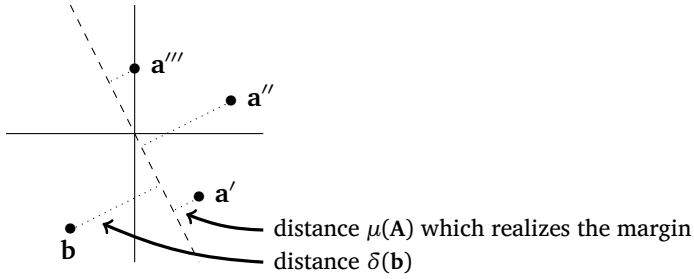
Theorem 5 from Subsection 4.2 provides the approximate expression (22) of the error bound for the HG error-driven learner in the noisy learning setting. The precise formulation of the error bound is provided in (34).

Theorem 5. *Consider the HG error-driven learner with the deterministic update condition (24a) and the original Perceptron reweighing rule (24b). Assume it is trained on a sequence of EWCs sampled from two EWC sets* $\mathbf{A}$ *and* $\mathbf{B}$*. The EWCs of* $\mathbf{A}$ *are called* pristine *because they are consistent with some HG grammar with margin* $\mu(\mathbf{A})$*. The EWCs of* $\mathbf{B}$ *are called* corrupted *because each of them is inconsistent with the EWCs in* $\mathbf{A}$*. Assume that the set* $\mathbf{A}$ *of pristine EWCs is finite and that the training sequence contains only a finite number of corrupted EWCs from* $\mathbf{B}$*. The number of errors made by the learner on this training sequence is at most*

$$(34) \quad \left(\frac{\rho(\mathbf{A} \cup \mathbf{B}) + \sqrt{\sum_{\mathbf{b}\in\mathbf{B}} n(\mathbf{b})\Big(\mu(\mathbf{A}) + \delta(\mathbf{b})\Big)^2}}{\mu(\mathbf{A})} \right)^2$$

where $\rho(\mathbf{A} \cup \mathbf{B})$ *is the radius of the pristine data* $\mathbf{A}$ *plus the corrupted data* $\mathbf{B}$*,* $\mu(\mathbf{A})$ *is the margin of the pristine data* $\mathbf{A}$*,* $n(\mathbf{b})$ *is the number of times*

Figure 2: Illustration of Theorem 5



that the corrupted piece of data $\mathbf{b}$ *has been fed to the learner in the training sequence, and* $\delta(\mathbf{b})$ *is the distance of the corrupted piece of data* $\mathbf{b}$ *from the decision surface corresponding to the weight vector which realizes the margin of the pristine data.*

The theorem can be illustrated as follows. Suppose that there are only $n = 2$ constraints and that the set of pristine EWCs is $\mathbf{A} = \{\mathbf{a}', \mathbf{a}'', \mathbf{a}'''\}$ plotted in Figure 2. The decision line which realizes the margin of these pristine data is represented by the dashed line. The margin is the distance $\mu(\mathbf{A})$ of the closest EWC $\mathbf{a}'$ from the dashed line. The EWC $\mathbf{b}$ is corrupted because inconsistent with the pristine data (it sits in the opposite half plane). The distance of this corrupted piece of data $\mathbf{b}$ from the decision line which realizes the margin is denoted by $\delta(\mathbf{b})$. The "quantity which depends on the corrupted data" mentioned in the approximate expression (22) of the error bound is thus the square root in the numerator of (34), namely the square root of the sum of the number $n(\mathbf{b})$ of times each corrupted piece of data $\mathbf{b}$ is fed to the learner, weighted by (the square of) the distance $\delta(\mathbf{b})$ plus the distance $\mu(\mathbf{A})$.

REFERENCES

Tamás Sándor BÍRÓ (2006), *Finding the right words: Implementing Optimality Theory with Simulated Annealing*, Ph.D. thesis, University of Groningen, available as ROA-896.

Hans-Dieter BLOCK (1962), The perceptron: A model of brain functioning, *Review of Modern Physics*, 34(1):123–135.

Paul BOERSMA (1997), How we learn variation, optionality and probability, in Rob VAN SON, editor, *Proceedings of the Institute of Phonetic Sciences (IFA) 21*, pp. 43–58, Institute of Phonetic Sciences, University of Amsterdam.

Paul BOERSMA (1998), *Functional Phonology*, Ph.D. thesis, University of Amsterdam, The Netherlands, holland Academic Graphics.

Paul BOERSMA and Bruce HAYES (2001), Empirical tests for the Gradual Learning Algorithm, *Linguistic Inquiry*, 32(1):45–86.

Paul BOERSMA and Joe PATER (to appear), Convergence properties of a gradual learner for Harmonic Grammar, in John MCCARTHY and Joe PATER, editors, *Harmonic Grammar and Harmonic Serialism*, Equinox Press.

Paul BOERSMA and Jan-Willem VAN LEUSSEN (2014), Fast evaluation and learning in multi-level parallel constraint grammars, University of Amsterdam.

Nicolò CESA-BIANCHI and Gábor LUGOSI (2006), *Prediction, learning, and games*, Cambridge University Press.

Andries W. COETZEE and Shigeto KAWAHARA (2013), Frequency biases in phonological variation, *Natural Language and Linguistic Theory*, 31(1):47–89.

Andries W. COETZEE and Joe PATER (2008), Weighted constraints and gradient restrictions on place co-occurrence in Muna and Arabic, *Natural Language and Linguistic Theory*, 26(2):289–337.

Andries W. COETZEE and Joe PATER (2011), The place of variation in phonological theory, in John GOLDSMITH, Jason RIGGLE, and Alan YU, editors, *Handbook of phonological theory*, pp. 401–434, Blackwell.

Nello CRISTIANINI and John SHAWE-TAYLOR (2000), *An introduction to Support Vector Machines and other kernel-based methods*, Cambridge University Press.

Robert FRANK and Shyam KAPUR (1996), On the use of triggers in parameter setting, *Linguistic Inquiry*, 27(4):623–660.

Yoav FREUND and Robert E. SCHAPIRE (1999), Large margin classification using the Perceptron algorithm, *Machine Learning*, 37(3):277–296.

Edward GIBSON and Kenneth WEXLER (1994), Triggers, *Linguistic Inquiry*, 25(3):407–454.

Bruce HAYES (2004), Phonological acquisition in Optimality Theory: The early stages, in René KAGER, Joe PATER, and Wim ZONNEVELD, editors, *Constraints in phonological acquisition*, pp. 158–203, Cambridge University Press.

Gaja JAROSZ (2013), Learning with hidden structure in Optimality Theory and Harmonic Grammar: Beyond Robust Interpretative Parsing, *Phonology*, 30(1):27–71.

Karen JESNEY and Anne-Michelle TESSIER (2011), Biases in Harmonic Grammar: the road to restrictive learning, *Natural Language and Linguistic Theory*, 29(1):251–290.

Frank KELLER (2000), *Gradience in grammar. Experimental and computational aspects of degrees of grammaticality*, Ph.D. thesis, University of Edinburgh, England.

Jyrki KIVINEN (2003), Online learning of linear classifiers, in Shahar MENDELSON and Alexander J. SMOLA, editors, *Advanced lectures on Machine Learning (LNAI 2600)*, pp. 235–257, Springer.

Jyrki KIVINEN, Manfred K. WARMUTH, and Peter AUER (1997), The Perceptron algorithm versus Winnow: linear versus logarithmic mistake bounds when few input variables are relevant, *Artificial Intelligence*, 97(1–2):325–343.

Norbert KLASNER and Hans-Ulrich SIMON (1995), From noise-free to noise-tolerant and from on-line to batch learning, in Wolfgang MAASS, editor, *Computational Learning Theory (COLT) 8*, pp. 250–257, ACM.

Gèraldine LEGENDRE, Yoshiro MIYATA, and Paul SMOLENSKY (1998a), Harmonic Grammar: A formal multi-level connectionist theory of linguistic well-formedness: An application, in Morton Ann GERNSBACHER and Sharon J. DERRY, editors, *Annual conference of the Cognitive Science Society 12*, pp. 884–891, Lawrence Erlbaum Associates.

Géraldine LEGENDRE, Yoshiro MIYATA, and Paul SMOLENSKY (1998b), Harmonic Grammar: A formal multi-level connectionist theory of linguistic well-formedness: Theoretical foundations, in Morton Ann GERNSBACHER and Sharon J. DERRY, editors, *Annual conference of the Cognitive Science Society 12*, pp. 388–395, Lawrence Erlbaum.

Gèraldine LEGENDRE, Antonella SORACE, and Paul SMOLENSKY (2006), The Optimality Theory/Harmonic Grammar connection, in Paul SMOLENSKY and Gèraldine LEGENDRE, editors, *The Harmonic Mind*, pp. 903–966, MIT Press.

Nick LITTLESTONE (1988), Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm, *Machine Learning*, 2(4):285–318.

Giorgio MAGRI (2015), Idempotency in Optimality Theory, manuscript.

Giorgio MAGRI (to appear), Error-driven learning in OT and HG: a comparison, *Phonology*.

Marvin MINSKY and Seymour PAPERT (1969), *Perceptrons: An introduction to Computational Geometry*, MIT Press.

Mehryar MOHRI and Afshin ROSTAMIZADEH (2013), Perceptron mistake bounds, arXiv:1305.0208.

Mehryar MOHRI, Afshin ROSTAMIZADEH, and Ameet TALWALKAR (2012), *Foundations of Machine Learning*, MIT Press.

Albert B. J. NOVIKOFF (1962), On convergence proofs on Perceptrons, in *Proceedings of the symposium on the mathematical theory of automata*, volume XII, pp. 615–622.

Joe PATER (2008), Gradual learning and convergence, *Linguistic Inquiry*, 39(2):334–345.

Alan PRINCE (2002), Entailed Ranking Arguments, ms., Rutgers University, New Brunswick, NJ. Rutgers Optimality Archive, ROA 500. Available at http://www.roa.rutgers.edu.

Alan PRINCE and Bruce TESAR (2004), Learning phonotactic distributions, in René KAGER, Joe PATER, and Wim ZONNEVELD, editors, *Constraints in phonological acquisition*, pp. 245–291, Cambridge University Press.

Frank ROSENBLATT (1958), The Perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review*, 65(6):386–408.

Frank ROSENBLATT (1962), *Principles of Neurodynamics*, Spartan.

Shai SHALEV-SHWARTZ and Yoram SINGER (2005), A new perspective on an old Perceptron algorithm, in Peter AUER and Ron MEIR, editors, *Conference on Computational Learning Theory (COLT) 18*, Lecture notes in Computer Science, pp. 264–278, Springer.

Paul SMOLENSKY and Gèraldine LEGENDRE (2006), *The Harmonic Mind*, MIT Press.

Kenneth WEXLER and Peter W. CULICOVER (1980), *Formal principles of language acquisition*, MIT Press, Cambridge, MA.

This work is licensed under the Creative Commons Attribution 3.0 Unported License.

`http://creativecommons.org/licenses/by/3.0/`