Online Learning of Task-specific Word Representations with a Joint Biconvex Passive-Aggressive Algorithm

Pascal Denis[1] **and Liva Ralaivola**[2]

[1]MAGNET, Inria Lille – Nord Europe , Villeneuve d'Ascq, France
pascal.denis@inria.fr
[2] QARMA, LIF, Aix-Marseille University, Marseille, France
liva.ralaivola@lif.univ-mrs.fr

Abstract

This paper presents a new, efficient method for learning task-specific word vectors using a variant of the Passive-Aggressive algorithm. Specifically, this algorithm learns a word embedding matrix in tandem with the classifier parameters in an online fashion, solving a biconvex constrained optimization at each iteration. We provide a theoretical analysis of this new algorithm in terms of regret bounds, and evaluate it on both synthetic data and NLP classification problems, including text classification and sentiment analysis. In the latter case, we compare various pre-trained word vectors to initialize our word embedding matrix, and show that the matrix learned by our algorithm vastly outperforms the initial matrix, with performance results comparable or above the state-of-the-art on these tasks.

1 Introduction

Recently, distributed word representations have become a crucial component of many natural language processing systems (Koo et al., 2008; Turian et al., 2010; Collobert et al., 2011). The main appeal of these word embeddings is two-fold: they can be derived directly from raw text data in an unsupervised or weakly-supervised manner, and their latent dimensions condense interesting distributional information about the words, thus allowing for better generalization while also mitigating the presence of rare and unseen terms. While there are now many different spectral, probabilistic, and deep neural approaches for building vectorial word representations, there is still no clear understanding as to which syntactic and semantic information they really capture and whether or how these representations really differ (Chen et al., 2013; Levy and Goldberg, 2014b; Schnabel et al., 2015). Also poorly understood is the relation between the word representations and the particular learning algorithm (e.g., whether linear or non-linear) that uses them as input (Wang and Manning, 2013).

What seems clear, however, is that there is no single best embedding and that their impact is very much task-dependent. This in turn raises the question of how to learn word representations that are adapted to a particular task and learning objective. Three different research routes have been explored towards learning task-specific word embeddings. A first approach (Collobert et al., 2011; Maas et al., 2011) is to learn the embeddings for the target problem jointly with additional unlabeled or (weakly-)labeled data in a semi-supervised or multi-task approach. While very effective, this joint training typically requires large amounts of data and often prohibitive processing times in the case of multi-layer neural networks (not to mention their lack of theoretical learning guarantees in part due to their strong non-convexity). Another approach consists in training word vectors using some existing algorithm as in like word2vec (Mikolov et al., 2013) in a way that exploits prior domain knowledge (e.g., by defining more informative, task-specific contexts) (Bansal et al., 2014; Levy and Goldberg, 2014a). In this case, there is still a need for additional weakly- or hand-labeled data, and there is no guarantee that the newly learned embeddings will indeed benefit the performance, as they are trained independently of the task objective. A third approach is to start with some existing pre-trained embeddings and fine-tune them to the task by integrating them in a joint learning objective either using backpropagation (Lebret et al., 2013) or regularized logistic regression (Labutov and Lipson, 2013). These

Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, pages 775–784,
Valencia, Spain, April 3-7, 2017.

approaches in effect hit a sweet spot by leveraging pre-trained embeddings and requiring no additional data or domain knowledge, while directly tying them to task learning objective.

Inspired by these latter approaches, we propose a new, online soft-margin classification algorithm, called Re-embedding Passive-Agressive (or RPA), that jointly learns an embedding matrix in tandem with the model parameters. As its name suggests, this algorithm generalizes the Passive-Aggressive (PA) algorithm of Crammer and Singer (2006) by allowing the data samples to be projected into the lower dimension space defined by the original embedding matrix. Our approach may be seen as extending the work of (Grandvalet and Canu, 2003) which addresses the problem of simultaneously learning the features and the weight vector of an SVM classifier. An important departure, beyond the online nature of RPA, is that it learns a projection matrix and not just a diagonal one (which is essentially what this earlier work does). Our approach and analysis are also related to (Blondel et al., 2014), which tackles non-negative matrix factorization with the PA philosophy.

The main contributions of this paper are as follows. First, we derive a new variant of the Passive-Aggressive algorithm able to jointly learn an embedding matrix along with the weight vector of the model (section 2). Second, we provide theoretical insights as to bound the cumulative squared loss of our learning procedure over any given sequence of examples—the results we give are actually related to a learning procedure that slighlty differs from the algorithm we introduce but that is more easily and compactly amenable to a theoretical study. Third, we further study the behavior of this algorithm on synthetic data (section 4) and we finally show that it performs well on five real-world NLP classification problems (section 5).

2 Algorithm

We consider the problem of learning a binary linear classification function $f_{\boldsymbol{\Phi},\boldsymbol{w}}$, parametrized by both a weight vector $\boldsymbol{w} \in \mathbb{R}^k$ and an embedding matrix $\boldsymbol{\Phi} \in \mathbb{R}^{k \times p}$ (typically, with $k \ll p$), which is defined as:

$$\begin{array}{rccl} f_{\boldsymbol{\Phi},\boldsymbol{w}} & : & \mathcal{X} \subset \mathbb{R}^p & \rightarrow & \{-1,+1\} \\ & & \boldsymbol{x} & \mapsto & \text{sign}(\langle \boldsymbol{w}, \boldsymbol{\Phi}\boldsymbol{x} \rangle) \end{array}$$

We aim at an online learning scenario, wherein both $\boldsymbol{w}$ and $\boldsymbol{\Phi}$ will be updated in a sequential fashion. Given a labeled data steam $S = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^T$, it seems relevant at each step to solve the following soft-margin constrained optimization problem:

$$\underset{\substack{\boldsymbol{w} \in \mathbb{R}^k \\ \boldsymbol{\Phi} \in \mathbb{R}^{k \times p}}}{\text{argmin}} \frac{1}{2} \|\boldsymbol{w} - \boldsymbol{w}_t\|_2^2 + \frac{\lambda}{2} \|\boldsymbol{\Phi} - \boldsymbol{\Phi}_t\|_F^2 + C\xi^2 \quad (1a)$$

$$\text{s.t. } \ell_t(\boldsymbol{w}; \boldsymbol{\Phi}; \boldsymbol{x}_t) \leq \xi \quad (1b)$$

where $\|\cdot\|_2$, $\|\cdot\|_F$ stand for the l_2 and Frobenius norms, respectively, and C controls the "aggressiveness" of the update (as larger C values imply updates that are directly proportional to the incurred loss). We define $\ell_t(\boldsymbol{w}; \boldsymbol{\Phi}; \boldsymbol{x}_t)$ as the hinge loss, that is:

$$\ell_t(\boldsymbol{w}; \boldsymbol{\Phi}; \boldsymbol{x}_t) \doteq \max(0, 1 - y_t \langle \boldsymbol{w}, \boldsymbol{\Phi}\boldsymbol{x}_t \rangle). \quad (2)$$

The optimization problem in (1) is reminiscent of the soft-margin Passive-Aggressive algorithm proposed in (Crammer et al., 2006) (specifically, PA-II), but both the objective and the constraint now include a term based on the embedding matrix $\boldsymbol{\Phi}$. The λ regularization parameter in the objective controls the allowed divergence in the embedding parameters between iterations.

Interestingly, the new objective remains convex, but the margin constraint doesn't as it involves a multiplicative term between the weight vector and the embedding matrix, making the overall problem bi-convex (Gorski et al., 2007). That is, the problem is convex in $\boldsymbol{w}$ for fixed values of $\boldsymbol{\Phi}$ and convex in $\boldsymbol{\Phi}$ for fixed values of $\boldsymbol{w}$. Incidentally, the formulation presented by (Labutov and Lipson, 2013) is also bi-convex (as it also involves a similar multiplicative term), although the authors proceed as if it were jointly convex (i.e., convex in both $\boldsymbol{w}$ *and* $\boldsymbol{\Phi}$). In order to solve this problem, we resort to an alternating update procedure which updates each set of parameters (i.e., either the weight vector or the embedding matrix) while holding the other fixed until some stopping criterion is met (in our case, the value of the objective doesn't change). As shown in Algorithm 1, this procedure allows us to compute closed-form updates similar to those of PA, and to make use of the same theoretical apparatus for analyzing RPA.

2.1 Unified Formalization

Suppose from now on that C and λ are fixed and so are w_t and $\boldsymbol{\Phi}_t$: this will allow us to drop the explicit dependence on these values and to keep

Algorithm 1 Re-embedding Passive-Aggressive

Require:

- $S = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^T$, a stream of data
- a base embedding $\boldsymbol{\Phi}_0$

Ensure:

- a classification vector $\boldsymbol{w}$
- a re-embedding matrix $\boldsymbol{\Phi}$

Initialize $\boldsymbol{w}_0$
$t \leftarrow 0$
repeat
$\boldsymbol{w}_{t+1}^0 \leftarrow \boldsymbol{w}_t$, $\boldsymbol{\Phi}_{t+1}^0 \leftarrow \boldsymbol{\Phi}_t$,
$n \leftarrow 1$
repeat

$$\boldsymbol{w}_{t+1}^{n+1} \leftarrow \boldsymbol{w}_{t+1}^n + \frac{\ell_t(\boldsymbol{w}_{t+1}^n; \boldsymbol{\Phi}_{t+1}^n)}{\|\boldsymbol{\Phi}_{t+1}^n \boldsymbol{x}_t\|_2^2 + \frac{1}{2C}} y_t \boldsymbol{x}_t$$

$$\boldsymbol{\Phi}_{t+1}^{n+1} \leftarrow \boldsymbol{\Phi}_{t+1}^n + \frac{\ell_t(\boldsymbol{w}_{t+1}^{n+1}; \boldsymbol{\Phi}_{t+1}^n)}{\left\|\boldsymbol{w}_{t+1}^{n+1}\right\|_2^2 \|\boldsymbol{x}_t\|_2^2 + \frac{\lambda}{2C}} y_t \boldsymbol{w}_{t+1}^{n+1} \boldsymbol{x}_t^\top.$$

with

$$\ell_t(\boldsymbol{w}; \boldsymbol{\Phi}) = \max(0, 1 - y_t \langle \boldsymbol{w}, \boldsymbol{\Phi} \boldsymbol{x}_t \rangle).$$

until $\boldsymbol{w}_{t+1}^n \to \boldsymbol{w}_{t+1}^\infty$ and $\boldsymbol{\Phi}_{t+1}^n \to \boldsymbol{\Phi}_{t+1}^\infty$
Update $\boldsymbol{w}_{t+1}$ and $\boldsymbol{\Phi}_{t+1}$:

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_{t+1}^\infty \qquad \boldsymbol{\Phi}_{t+1} \leftarrow \boldsymbol{\Phi}_{t+1}^\infty$$

until some stopping criterion is met
return $\boldsymbol{w}_t, \boldsymbol{\Phi}_t$

the notation light. Let Q_t be defined as:

$$Q_t(\boldsymbol{w}, \boldsymbol{\Phi}, \xi) \doteq \frac{1}{2} \|\boldsymbol{w} - \boldsymbol{w}_t\|_2^2 + \frac{\lambda}{2} \|\boldsymbol{\Phi} - \boldsymbol{\Phi}_t\|_F^2 + C\xi^2. \tag{3}$$

and margin q_t be defined as:

$$q_t(\boldsymbol{w}, \boldsymbol{\Phi}) \doteq 1 - \langle \boldsymbol{w}, y_t \boldsymbol{\Phi} \boldsymbol{x}_t \rangle \tag{4a}$$

$$= 1 - \left\langle \boldsymbol{\Phi}, y_t \boldsymbol{w} \boldsymbol{x}_t^\top \right\rangle_F. \tag{4b}$$

Here $\langle \cdot, \cdot \rangle_F$ denote the Frobenius inner product. We have purposely provided the two equivalent forms (4a) and (4b) to emphasize the (syntactic) exchangeability of $\boldsymbol{w}$ and $\boldsymbol{\Phi}$. As we shall see, this is going to be essential to derive an alternating update procedure—which alternates the updates with respect to $\boldsymbol{w}$ and $\boldsymbol{\Phi}$—in a compact way.

Given Q_t and q_t, we are now interested in solving the following optimization problem:

$$\boldsymbol{w}_{t+1}, \boldsymbol{\Phi}_{t+1}, \xi_{t+1} = \operatorname*{argmin}_{\boldsymbol{w}, \boldsymbol{\Phi}, \xi} Q_t(\boldsymbol{w}, \boldsymbol{\Phi}, \xi) \tag{5a}$$

$$\text{s.t. } q_t(\boldsymbol{w}, \boldsymbol{\Phi}) \leq \xi. \tag{5b}$$

2.2 Bi-convexity

It turns out that problem (5) is a bi-convex optimization problem: it is indeed straightforward to observe that if $\boldsymbol{\Phi}$ is fixed then the problem is convex in $(\boldsymbol{w}, \xi)$—it is the classical passive-aggressive II optimization problem—and if $\boldsymbol{w}$ is fixed then the problem is convex in $(\boldsymbol{\Phi}, \xi)$. If there exist theoretical results on the solving of bi-convex optimization problems, the machinery to use pertains to combinatorial optimization, which might be too expensive. In addition, computing the solution of (5) would drive us away from the spirit of passive-aggressive learning which rely on cheap and statistically meaningful (from the mistake-bound perspective) updates. This is the reason why we propose to resort to an alternate online procedure to solve a proxy of (5).

2.3 An Alternating Online Procedure

Instead of tackling problem (5) directly we propose to solve, at each time t, either of:

$$\boldsymbol{w}_{t+1}, \xi_{t+1} = \operatorname*{argmin}_{\boldsymbol{w}, \xi} Q_t(\boldsymbol{w}, \boldsymbol{\Phi}_t, \xi) \text{ s.t. } q_t(\boldsymbol{w}, \boldsymbol{\Phi}_t) \leq \xi, \tag{6}$$

$$\boldsymbol{\Phi}_{t+1}, \xi_{t+1} = \operatorname*{argmin}_{\boldsymbol{\Phi}, \xi} Q_t(\boldsymbol{w}_t, \boldsymbol{\Phi}, \xi) \text{ s.t. } q_t(\boldsymbol{w}_t, \boldsymbol{\Phi}) \leq \xi. \tag{7}$$

This means that the optimization is performed with either $\boldsymbol{\Phi}$ fixed to $\boldsymbol{\Phi}_t$ or $\boldsymbol{w}$ fixed to $\boldsymbol{w}_t$.

Informally, the iterative Algorithm 1, resulting from this alternate scheme, will solve at each round a simple constrained optimization problem, in which the objective is to minimize the squared Euclidean distances between the new weight vector (resp. the new embedding matrix) and the current one, while making sure that both sets of parameters achieve a correct prediction with a sufficiently high margin. Note that one may recover the standard passive-aggressive algorithm by simply fixing the embedding matrix to the identity matrix. Also note that if the right stopping criteria are retained, Algorithm 1 is guaranteed to converge to a local optimum of (5) (see (Gorski et al., 2007)).

When fully developed, problems (6)-(7) respectively write as:

$$\boldsymbol{w}_{t+1}, \xi_{t+1} = \operatorname*{argmin}_{\boldsymbol{w}, \xi} \frac{1}{2} \|\boldsymbol{w} - \boldsymbol{w}_t\|_2^2 + C\xi^2$$
$$\text{s.t. } 1 - \langle \boldsymbol{w}, y_t \boldsymbol{\Phi}_t \boldsymbol{x}_t \rangle \leq \xi,$$

or

$$\boldsymbol{\Phi}_{t+1}, \xi_{t+1} = \underset{\boldsymbol{\Phi}, \xi}{\operatorname{argmin}} \ \frac{\lambda}{2}\|\boldsymbol{\Phi} - \boldsymbol{\Phi}_t\|_F^2 + C\xi^2$$
$$\text{s.t. } 1 - \left\langle \boldsymbol{\Phi}, y_t \boldsymbol{w}_t \boldsymbol{x}_t^\top \right\rangle_F \leq \xi.$$

Using the equivalence in (4), both problems can be seen as special instances of the generic problem:

$$\boldsymbol{u}^*, \xi^* = \underset{\boldsymbol{u}, \xi}{\operatorname{argmin}} \ \frac{\lambda}{2}\|\boldsymbol{u} - \boldsymbol{u}_t\|_\star^2 + C\xi^2 \tag{8a}$$
$$\text{s.t. } 1 - \langle \boldsymbol{u}, \boldsymbol{v}_t \rangle_\star \leq \xi. \tag{8b}$$

where $\|\cdot\|_\star$ and $\langle \cdot, \cdot \rangle_\star$ are generalized versions of the l_2 norm and the inner product, respectively.

This is a convex optimization problem that can be readily solved using classical tools from convex optimization to give:

$$\boldsymbol{u}^* = \boldsymbol{u}_t + \tau_{\boldsymbol{u}} \boldsymbol{v}_t, \text{ with } \tau_{\boldsymbol{u}} = \frac{\max(0, 1 - \langle \boldsymbol{u}_t, \boldsymbol{v}_t \rangle_\star)}{\|\boldsymbol{v}_t\|_\star^2 + \frac{\lambda}{2C}}, \tag{9}$$

which comes from the following proposition.

Proposition 1. *The solution of* (8) *is* (9).

Proof. The Lagrangian associated with the problem is given by:

$$\mathcal{L}(\boldsymbol{u}, \xi, \tau) = \frac{\lambda}{2}\|\boldsymbol{u} - \boldsymbol{u}_t\|_\star^2 + C\xi^2 + \tau\left(1 - \xi - \langle \boldsymbol{u}, \boldsymbol{v}_t \rangle_\star\right) \tag{10}$$

with $\tau > 0$.

Necessary conditions for optimality are $\nabla_{\boldsymbol{u}} \mathcal{L} = \mathbf{0}$, and $\nabla_\xi \mathcal{L} = 0$, which imply $\boldsymbol{u} = \boldsymbol{u}_t + \frac{\tau}{\lambda}\boldsymbol{v}_t$, and $\xi = \frac{\tau}{2C}$. Using this in (10) gives the function:

$$g(\tau) = \frac{\tau^2 \|\boldsymbol{v}_t\|_\star^2}{2\lambda} + \frac{\tau^2}{4C} + \tau - \frac{\tau^2}{2C} - \tau \langle \boldsymbol{u}_t, \boldsymbol{v}_t \rangle_\star - \frac{\tau^2 \|\boldsymbol{v}_t\|_\star^2}{\lambda},$$

which is maximized with respect to τ when $g'(\tau) = 0$, i.e.:

$$\left(\frac{\|\boldsymbol{v}_t\|_\star^2}{\lambda} - \frac{1}{2C} - \frac{2\|\boldsymbol{v}_t\|_\star^2}{\lambda}\right)\tau + 1 - \langle \boldsymbol{u}_t, \boldsymbol{v}_t \rangle_\star = 0.$$

Taking into account the constraint $\tau \geq 0$, the maximum of g is attained at $\tilde{\tau}$ with:

$$\tilde{\tau} = \frac{\lambda \max(0, 1 - \langle \boldsymbol{u}_t, \boldsymbol{v}_t \rangle_\star)}{\|\boldsymbol{v}_t\|_\star^2 + \frac{\lambda}{2C}},$$

which, setting $\tau_{\boldsymbol{u}} = \tilde{\tau}/\lambda$ gives (9). □

The previous proposition allows us to readily have the solutions of (6) and (7) as follows.

Proposition 2. *The updates induced by the solutions of* (6) *and* (7) *are respectively given by:*

$$\boldsymbol{w}^* = \boldsymbol{w}_t + \tau_{\boldsymbol{w}} y_t \boldsymbol{x}_t, \text{ with } \tau_{\boldsymbol{w}} = \frac{\ell_t(\boldsymbol{w}_t; \boldsymbol{\Phi}_t)}{\|\boldsymbol{\Phi}_t \boldsymbol{x}_t\|_2^2 + \frac{1}{2C}} \tag{11}$$

$$\boldsymbol{\Phi}^* = \boldsymbol{\Phi}_t + \tau_{\boldsymbol{\Phi}} y_t \boldsymbol{w}_t \boldsymbol{x}_t^\top, \text{ with } \tau_{\boldsymbol{\Phi}} = \frac{\ell_t(\boldsymbol{w}_t; \boldsymbol{\Phi}_t)}{\|\boldsymbol{w}_t\|_2^2 \|\boldsymbol{x}_t\|_2^2 + \frac{\lambda}{2C}}, \tag{12}$$

where $\ell_t(\boldsymbol{w}; \boldsymbol{\Phi}) = \max(0, q_t(\boldsymbol{w}, \boldsymbol{\Phi}))$.

Proof. Just instantiate the results of Proposition 1 with $(\boldsymbol{u}, \boldsymbol{v}_t) = (\boldsymbol{w}, y_t \boldsymbol{\Phi}_t \boldsymbol{x}_t)$ for (11) and $(\boldsymbol{u}, \boldsymbol{v}_t) = (\boldsymbol{\Phi}, y_t \boldsymbol{x}_t \boldsymbol{w}_t^\top)$ for (12). □

Remark 1 (Hard-margin case)**.** *Note that the hard-margin version of the previous problem:*

$$\underset{\substack{\boldsymbol{w} \in \mathbb{R}^k \\ \boldsymbol{\Phi} \in \mathbb{R}^{k \times p}}}{\operatorname{argmin}} \ \frac{1}{2}\|\boldsymbol{w} - \boldsymbol{w}_t\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\Phi} - \boldsymbol{\Phi}_t\|_F^2 \tag{13}$$
$$\text{s.t. } 1 - y_t \langle \boldsymbol{w}, \boldsymbol{\Phi} \boldsymbol{x}_t \rangle \leq 0 \tag{14}$$

is degenerate from the alternated optimization point of view. It suffices to observe that the updates entailed by the hard-margin problem correspond to (9) *with C set to ∞; if it happens that either $\boldsymbol{\Phi}_0 = \mathbf{0}$ or $\boldsymbol{w}_0 = \mathbf{0}$, then one of the optimization problems (wrt to $\boldsymbol{w}$ or $\boldsymbol{\Phi}$) has no solution.*

3 Analysis

Using the same technical tools as in (Crammer et al., 2006), and the unified formalization of section 2, we have the following result.

Proposition 3. *Suppose that problem* (8) *is iteratively solved for a sequence of vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_T$ to give $\boldsymbol{u}_2, \ldots, \boldsymbol{u}_{T+1}$, $\boldsymbol{u}_1$ being given. Suppose that, at each time step, $\|\boldsymbol{v}_t\|_\star \leq R$, for some $R > 0$. Let $\boldsymbol{u}^*$ be an arbitrary vector living in the same space as $\boldsymbol{u}_1$. The following result holds*

$$\sum_{t=1}^T \ell_t^2 \leq \left(R^2 + \frac{\lambda}{2C}\right)\left(\|\boldsymbol{u}^* - \boldsymbol{u}_1\|_\star^2 + \frac{2C}{\lambda}\sum_{t=1}^T (\ell_t^*)^2\right) \tag{15}$$

where

$$\ell_t = \max(0, 1 - \langle \boldsymbol{u}_t, \boldsymbol{v}_t \rangle_\star), \text{ and } \ell_t^* = \max(0, 1 - \langle \boldsymbol{u}^*, \boldsymbol{v}_t \rangle_\star). \tag{16}$$

This proposition and the accompanying lemmas are simply a variation of the results of (Crammer et al., 2006), with the addition that they are based on the generic problem (8). The loss bound applies for a version of Algorithm 1 where one of the parameters, either the weight vector or the re-embedding matrix, is kept fixed for some time.

Proposition 3 makes use of the following lemma (see Lemma 1 in (Crammer et al., 2006)):

Lemma 1. *Suppose that problem* (8) *is iteratively solved for a sequence of vectors* $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_T$ *to give* $\boldsymbol{u}_2, \ldots, \boldsymbol{u}_{T+1}$, $\boldsymbol{u}_1$ *being given. The following holds:*

$$\sum_{t=1}^{T} \tau_{\boldsymbol{u}}(2\ell_t - \tau_{\boldsymbol{u}} \|\boldsymbol{v}_t\|_\star^2 - 2\ell_t^*) \leq \|\boldsymbol{u}^* - \boldsymbol{u}_1\|_\star^2 . \quad (17)$$

Proof. As in (Crammer et al., 2006), simply set $\Delta_t = \|\boldsymbol{u}_t - \boldsymbol{u}^*\|_\star^2 - \|\boldsymbol{u}_{t+1} - \boldsymbol{u}^*\|_\star^2$ and bound $\sum_{\tau=1}^{T} \Delta_t$ from above and below. For the upper bound: $\sum_{\tau=1}^{T} \Delta_t = \|\boldsymbol{u}_1 - \boldsymbol{u}^*\|_\star^2 - \|\boldsymbol{u}_{T+1} - \boldsymbol{u}^*\|_\star^2 \leq \|\boldsymbol{u}_1 - \boldsymbol{u}^*\|_\star^2$.

For the lower bound, we focus on the nontrivial situation where $\ell_t > 0$, otherwise $\Delta_t = 0$ (i.e. no update is made) and the bounding is straightforward. Making use of the value of $\tau_{\boldsymbol{u}}$:

$$\begin{aligned}
\Delta_t &= \|\boldsymbol{u}_t - \boldsymbol{u}^*\|_\star^2 - \|\boldsymbol{u}_{t+1} - \boldsymbol{u}^*\|_\star^2 \\
&= \|\boldsymbol{u}_t - \boldsymbol{u}^*\|_\star^2 - \|\boldsymbol{u}_t + \tau_{\boldsymbol{u}} \boldsymbol{v}_t - \boldsymbol{u}^*\|_\star^2 \qquad \text{(see (8))} \\
&= -2\tau_{\boldsymbol{u}} \langle \boldsymbol{u}_t - \boldsymbol{u}^*, \boldsymbol{v}_t \rangle_\star - \tau_{\boldsymbol{u}}^2 \|\boldsymbol{v}_t\|_\star^2 \\
&= -2\tau_{\boldsymbol{u}} \langle \boldsymbol{u}_t, \boldsymbol{v}_t \rangle_\star + 2\tau_{\boldsymbol{u}} \langle \boldsymbol{u}^*, \boldsymbol{v}_t \rangle_\star - \tau_{\boldsymbol{u}}^2 \|\boldsymbol{v}_t\|_\star^2 .
\end{aligned}$$

Since $\ell_t > 0$, then $\langle \boldsymbol{u}_t, \boldsymbol{v}_t \rangle_\star = 1 - \ell_t$; also, by the definition of ℓ_t^* (see (16)), $\ell_t^* \geq 1 - \langle \boldsymbol{u}^*, \boldsymbol{v}_t \rangle_\star$, or $\langle \boldsymbol{u}^*, \boldsymbol{v}_t \rangle_\star \geq 1 - \ell_t^*$. This readily gives

$$\begin{aligned}
\Delta_t &\geq -2\tau_{\boldsymbol{u}}(1 - \ell_t) + 2\tau_{\boldsymbol{u}}(1 - \ell_t^*) - \tau_{\boldsymbol{u}}^2 \|\boldsymbol{v}_t\|_\star^2 \\
&= 2\tau_{\boldsymbol{u}}\ell_t - 2\tau_{\boldsymbol{u}}\ell_t^* - \tau_{\boldsymbol{u}}^2 \|\boldsymbol{v}_t\|_\star^2 ,
\end{aligned}$$

hence the targetted lower bound and, in turn, (17). □

Proof of Proposition 3. As for the proof of Proposition 3, it suffices, again, to follow the steps given in (Crammer et al., 2006), but this time, for the proof of Theorem 5.

Note that for any $\beta \neq 0$, $(\beta\tau_{\boldsymbol{u}} - \ell_t^*/\beta)^2$ is well-defined and nonnegative and thus:

$$\begin{aligned}
&\|\boldsymbol{u}^* - \boldsymbol{u}_1\|_\star^2 \\
&\geq \sum_{t=1}^{T} \left[\tau_{\boldsymbol{u}}(2\ell_t - \tau_{\boldsymbol{u}} \|\boldsymbol{v}_t\|_\star^2 - 2\ell_t^*) - (\beta\tau_{\boldsymbol{u}} - \ell_t^*/\beta)^2\right] \\
&= \sum_{t=1}^{T} (2\tau_{\boldsymbol{u}}\ell_t - \tau_{\boldsymbol{u}}^2(\|\boldsymbol{v}_t\|_\star^2 + \beta^2) - 2\tau_{\boldsymbol{u}}\ell_t^2 \\
&\quad + 2\tau_{\boldsymbol{u}}\ell_t^2 - (\ell_t^*)^2/\beta^2) \\
&= \sum_{t=1}^{T} (2\tau_{\boldsymbol{u}}\ell_t - \tau_{\boldsymbol{u}}^2(\|\boldsymbol{v}_t\|_\star^2 + \beta^2) - (\ell_t^*)^2/\beta^2).
\end{aligned}$$

Setting $\beta = \sqrt{\lambda/2C}$ and using $\tau_{\boldsymbol{u}} = \ell_t/(\|\boldsymbol{v}\|_\star^2 + \frac{\lambda}{2C})$ (see (9)) gives

$$\begin{aligned}
&\|\boldsymbol{u}^* - \boldsymbol{u}_1\|_\star^2 \\
&\geq \sum_{t=1}^{T} \left(2\tau_{\boldsymbol{u}}\ell_t - \tau_{\boldsymbol{u}}^2 \left(\|\boldsymbol{v}_t\|_\star^2 + \frac{\lambda}{2C}\right) - \frac{2C}{\lambda}(\ell_t^*)^2\right), \\
&= \sum_{t=1}^{T} \left(\frac{\ell_t^2}{\|\boldsymbol{v}_t\|_\star^2 + \frac{\lambda}{2C}} - \frac{2C}{\lambda}(\ell_t^*)^2\right).
\end{aligned}$$

Using the assumption that $\|\boldsymbol{v}_t\|_\star \leq R$ fo all t and rearranging terms concludes the proof. □

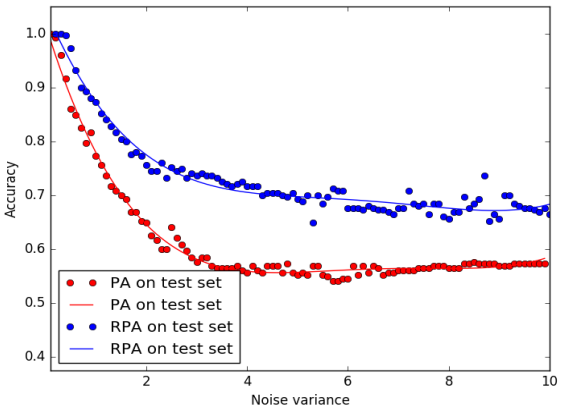

Figure 1: Accuracy rates for PA and RPA as a function of the variance of the Gaussian noise added to the "true" embedding Φ. The observed X matrix is $n = 500 \times p = 1000$.

The result given in Proposition 3 bounds the cumulative loss of the learning procedure when one of the parameters, either $\boldsymbol{\Phi}$ or $\boldsymbol{w}$, is fixed and the other is the optimization variable. Therefore, it does not directly capture the behavior of Algorithm 1, which alternates between the updates of $\boldsymbol{\Phi}$ and $\boldsymbol{w}$. A proper analysis of Algorithm 1 would require a refinement of Lemma 1 which, to our understanding, would be the core of a new result. This is a problem we intend to put our energy on in the near future, as an extension to this work.

4 Experiments on Synthetic Data

In order to better understand and validate the RPA algorithm, we first conducted some synthetic experiments. Specifically, we simulated a high-dimensional matrix $X \in \mathbb{R}^{n \times p}$, with n data samples realizing p "words" and $p \gg n$, using the following generative model:

$$X = Z\tilde{\Phi}, \text{where } \tilde{\Phi} = \Phi + \mathcal{E}$$

That is, each p-dimensional data point x_i was generated from a hidden lower k-dimensional z_i and

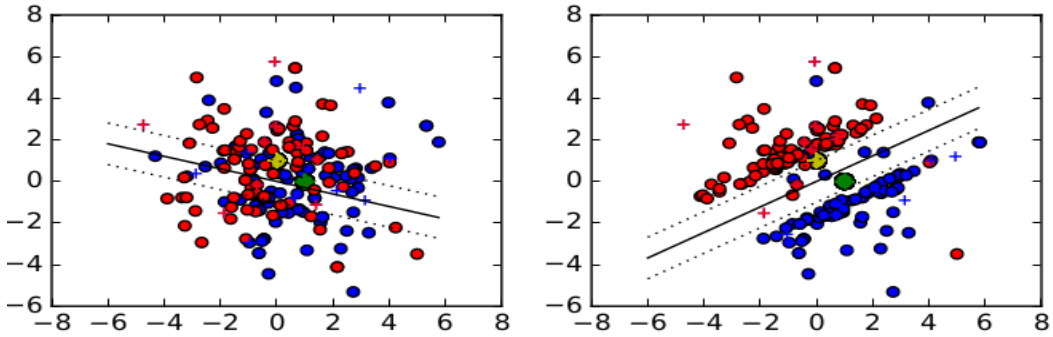

Figure 2: Hyper-plane learned by PA on $X\tilde{\Phi}^\top$ (left pane) compared to hyper-plane and data representations learned by RPA (right pane). X is $n = 200 \times p = 500$, and noise $\sigma = 2$. Training and test points appear as circles or plus marks, respectively. RPA's hyper-parameters are set to $C = 100$ and $\lambda = .5$.

an embedding matrix $\Phi \in \mathbb{R}^{k \times p}$, mapping k latent concepts to the p observed words. For simplicity, we assume that: (i) there are only two concepts (i.e., $k = 2$), (ii) each data point realizes a single concept (i.e., each x_i is a p-dimensional indicator vector), (iii) each concept is equally represented in the data (with $n/2$ data points), and (iv) each concept deterministically signals a class label, either -1 or $+1$. Recovering Z and predicting the z_i's labels is trivial if one is given X and the true embedding Φ, so we added Gaussian noise $\varepsilon_i \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbb{I}_p)$ to each ϕ_i. The resulting, observed noisy matrix is denoted $\tilde{\Phi}$.

Given this setting, the goal of the RPA is to learn a set of classification parameters $\boldsymbol{w} \in \mathbb{R}^k$ in the latent space and to "de-noise" the observed embedding matrix $\tilde{\Phi}$ by exploiting the labeled data. We are interested in comparing the RPA with a regular PA that directly learns from the noisy data $X\tilde{\Phi}^\top$. The outcome of this comparison is plotted in Figure 1. For this experiment, we randomly splitted the X data according to 80/10/10 for train/dev/test and considered increasing noise variance from 0 to 10 by increment of 0.1. Each dot in Figure 1 corresponds to the average accuracy over 10 seperate, random initializations of the embedding matrix at a particular noise level. Hyper-parameters were optimized using a grid search on the dev set for both the PA and RPA.[1]

As shown in Figure 1, the PA's accuracy quickly drops to levels that are only slightly above chance, while the RPA manages to maintain an accuracy close to .7 even with large noise. This indicates that the RPA is able to recover some of the structure in the embedding matrix. This behavior is also illustrated in Figure 2, wherein the two hidden concepts appear in yellow and green. While the standard PA learns a very bad hyper-plane, which fails to separate the two concepts, the RPA learns a much better hyper-plane. Interestingly, most of the data points appear to have been projected on the margins of the hyper-plane.

5 Experiments on NLP tasks

This section assesses the effectiveness of RPA on several text classification tasks.

5.1 Evaluation Datasets

We consider five different classification tasks which are concisely summarized in Table 1.

20 Newsgroups Our first three text classification tasks from this dataset[2] consists in categorizing documents into two related sub-topics: (i) **Comp.**: IBM vs. Mac, (ii) **Religion**: atheism vs. christian, and (iii) **Sports**: baseball vs. hockey.

IMDB Movie Review This movie dataset[3] was introduced by (Maas et al., 2011) for sentiment analysis, and contains $50,000$ reviews, divided into a balanced set of highly positive (7 stars out of 10 or more) and negative scores (4 stars or less).

TREC Question Classification This dataset[4] (Li and Roth, 2002) involves six question types: abbreviation, description, entity, human, location, and number.

[1] We use $\{1, 5, 10, 50\}$ for the number iterations, C's values were $\{.1, .5, 1.0, 2.0, 10.0\}$, and λ was set to 10^k, with $k \in \{-2, -1, 0, 1, 2\}$.

[2] qwone.com/~jason/20Newsgroups

[3] ai.stanford.edu/~amaas/data/sentiment

[4] cogcomp.cs.illinois.edu/Data/QA/QC

Name	lab.	examples		Voc. size	
		Train	Test	All	$fr > 1$
comp	2	1 168	777	30 292	15 768
rel.	2	1 079	717	32 361	18 594
sports	2	1 197	796	33 932	19 812
IMDB	2	$25k$	$25k$	172 001	86 361
TREC	6	5 452	500	9 775	3 771

Table 1: Number of labels, samples, vocabulary sizes (incl. non-hapax words) per dataset.

Word Embedding					
CnW	GV6	GV840	HPCA	HLBL	SkGr
40.41	27.18	20.26	32.20	40.44	32.19
25.45	13.07	8.95	16.90	25.39	17.33
35.29	19.01	13.96	29.97	35.21	30.66
39.79	12.67	4.93	22.15	39.63	16.79
3.42	0.61	0.40	1.38	3.63	4.51

Table 2: Out-of-vocabulary rates for non-hapax words in each dataset-embedding pair.

5.2 Preprocessing and Document Vectors

All datasets were pre-preprocessed with the Stanford tokenizer[5], except for the TREC corpus which comes pre-tokenized. Case was left intact unless used in conjunction with word embeddings that assume down-casing (see below).

For constructing document or sentence vectors, we used a simple 0-1 bag-of-word model, simply summing over the word vectors of occurring tokens, followed by L2-normalization of the resulting vector in order to avoid document/sentence length effects. For each dataset, we restricted the vocabulary to non-hapax words (i.e., words occurring more than once). Words unknown to the embedding were mapped to zero vectors.

5.3 Initial Word Vector Representations

Five publicly available word vectors were used to define initial embedding matrices in the RPA. The coverage of the different embeddings wrt each dataset vocabulary is reported in Table 2.

CnW These word vectors were induced using the neural language model of (Collobert and Weston, 2008) re-implemented by (Turian et al., 2010).[6] They were trained on 63M word news corpus, covering $268,810$ word forms (intact case), with 50, 100 or 200 dimensions for each word.

HLBL These were obtained using the probabilistic Hierarchical log-bilinear language model of (Mnih and Hinton, 2009), again re-implemented by (Turian et al., 2010) with 50 or 100 dimensions. They cover $246,122$ word forms.

HPCA (Lebret and Collobert, 2014) present a variant of Principal Component Analysis, Hellinger PCA, for learning spectral word vectors. These were trained over 1.62B words from Wikipedia, RCV1, and WSJ with all words lowercased, and digits mapped to a special symbol. Vocabulary is restricted to words occurring 100 times or more (hence, a total of $178,080$ words). These come in 50, 100 and 200 dimensions.[7]

GloVe These global word vectors are trained using a log-bilinear regression model on aggregated global word co-occurrence statistics (Pennington et al., 2014). We use two different releases:[8] (i) **GV6B** trained on Wikipedia 2014 and Gigaword 5 (amounting to 6B down-cased words and a vocabulary of 400k) with vectors in 50, 100, 200, or 300 dimensions, and (ii) **GV840B** trained over 840B uncased words (a 2.2M vocabulary) with vectors of length 300.

SkGr Finally, we use word vectors pre-trained with the skip-gram neural network model of (Mikolov et al., 2013): each word's Huffman code is fed to a log-linear classifier with a continuous projection layer that predicts context words within a specified window. The embeddings were trained on a 100B word corpus of Google news data (a $3M$ vocabulary) and are of length 300.[9]

rand In addition to these pre-trained word representations, we also use random vectors of lengths 50, 100, and 200 as a baseline embedding. Specifically, each component of these word vector is uniformly distributed on the unit interval $(-1, 1)$.

5.4 Settings

The hyperparameters of the model, C, λ, and the number it of iterations over the training set, were estimated using a grid search over $C = 10^{k \in \{-6,-4,-2,0,2,4,6\}}$, $\lambda = 10^{l \in \{-3,-2,-1,0,1,2,3\}}$, and $it \in \{1, 5, 10\}$ in a 10-fold cross-validation

[5] nlp.stanford.edu/software/tokenizer.shtml

[6] metaoptimize.com/projects/wordreprs

[7] lebret.ch/words

[8] nlp.stanford.edu/projects/glove

[9] code.google.com/archive/p/word2vec

Emb/Task		comp		religion		sports		trec		imdb		Average	
Method	Size	PA	RPA	PA	RPA	PA	RPA	PA	RPA	PA	RPA	PA	RPA
rand	50	54.57	87.13	60.67	91.91	63.07	92.34	56.80	88.40	61.47	86.47	59.32	89.25
	100	62.03	87.39	66.95	92.19	65.58	93.22	68.00	88.20	65.07	86.56	65.53	89.51
	200	65.51	87.64	73.22	92.19	71.11	92.96	70.00	88.20	69.10	86.58	69.79	89.51
CnW	50	50.32	85.97	55.09	91.91	56.91	93.72	69.40	87.20	58.68	86.54	58.08	89.07
	100	49.29	87.13	47.56	91.63	58.54	93.72	69.20	87.80	65.04	86.51	57.93	89.36
	200	50.32	87.00	53.14	91.77	63.94	93.72	75.80	87.60	68.21	86.64	62.28	89.35
HLBL	50	51.99	87.13	68.62	91.35	61.31	93.72	66.80	88.80	67.30	86.70	63.20	89.54
	100	53.54	86.74	65.83	91.77	63.07	93.72	75.60	88.20	72.41	86.68	66.09	89.42
HPCA	50	50.45	86.87	50.77	91.63	64.20	93.34	66.00	88.60	62.78	80.56	58.84	88.20
	100	50.45	86.87	47.98	91.63	64.57	93.72	72.00	88.40	64.25	85.33	59.85	89.19
	200	50.45	86.87	48.12	91.91	62.56	93.59	78.40	88.00	64.75	85.84	60.86	89.24
GV6B	50	55.21	86.87	75.59	91.91	86.68	95.23	65.80	89.00	75.12	86.41	71.68	89.88
	100	58.17	86.87	76.43	91.63	90.33	96.48	70.40	89.00	78.72	86.50	74.81	90.10
	200	70.40	86.87	73.22	91.63	93.34	97.11	76.60	89.00	81.55	86.57	79.02	90.24
	300	76.58	87.13	79.92	91.77	94.97	97.74	78.60	88.60	82.41	86.41	82.50	90.33
GV840B	300	75.80	87.13	88.15	92.05	88.69	96.23	77.20	89.20	84.17	86.46	82.80	90.21
SkGr	300	70.79	87.39	88.01	92.05	91.96	97.61	84.00	90.60	83.39	88.52	83.63	91.23
one-hot		87.26		91.91		93.47		88.00		88.29		89.786	

Table 3: Accuracy results on our five datasets for the Re-embedding Passive-Aggressive (RPA) against standard PA-II (PA). For each task, the best results for each embedding method (across different dimensions) has been greyed out, while the overall highest accuracy score has been underlined.

over the training data. For the alternating online procedure, we used the difference between the objective values from one iteration to the next for defining the stopping criterion, with maximum number of iterations of 50. In practice, we found that the search often converged much few iterations. The multi-class classifier used for the TREC dataset was obtained by training the RPA in simple One-versus-All fashion, thus learning one embedding matrix per class.[10] For datasets with label imbalance, we set a different C parameter for each class, re-weighting it in proportion to the inverse frequency of the class.

5.5 Results

Table 3 summarizes accuracy results for the RPA against those obtained by a PA trained with fixed pre-trained embeddings. The first thing to notice is that the RPA delivers massive accuracy improvements over the vanilla PA across datasets and embedding types and sizes, thus showing that the RPA is able to learn word representations that are better tailored to each problem. On average, accuracy gains are between 22% and 31% for CnW, HLBL, and HPCA. Sizable improvements, ranging from 8% and 18%, are also found for the better performing GV6B, GV840B, and SkGr. Second, RPA is able to outperform on all five datasets the strong baseline provided by the one-hot version of PA trained on the original high-dimensional space, with some substantial gains on *sports* and *trec*.

Overall, the best scores are obtained with the re-embedded SkGr vectors, which yield the best average accuracy, and outperform all the other configurations on two of the five datasets (*trec* and *imdb*). GV6B (dimension 300) has the second best average scores, outperforming all the other configurations on *sports*. Interestingly, embeddings learned from random vectors achieve performance that are often on a par or higher than those given by HLBL, HPCA or CnW initializations. They actually yield the best performance for the two remaining datasets: *comp* and *religion*. On these tasks, RPA does not seem to benefit from the information contained in the pre-trained embeddings, or their coverage is not just good enough.

For both PA and RPA, performance appear to be positively correlated with embedding coverage: embeddings with lower OOV rates generally perform better those with more missing words. The correlation is only partial, since GV840B do not yield gains compared to GV6B and SkGr despite its better word coverage. Also, SkGr largely outperforms HPCA although they have similar OOV

[10] More interesting configurations (e.g., a single embedding matrix shared across classes), are left for future work.

Emb/Task		comp		religion		sports		trec		imdb		Average	
Method	Size	PA	RPA	PA	RPA	PA	RPA	PA	RPA	PA	RPA	PA	RPA
rand	50	56.76	84.04	59.55	90.24	64.20	90.08	53.20	83.60	60.97	85.74	58.94	86.74
	100	63.06	84.17	67.50	91.35	65.58	90.95	61.20	83.00	64.12	85.90	64.29	87.07
	200	64.86	85.07	71.27	91.07	68.59	90.95	59.60	83.40	67.84	85.75	66.43	87.25
cnw	50	50.32	84.81	55.51	90.93	50.63	91.08	61.20	84.20	50.86	84.82	53.70	87.17
	100	50.71	84.81	55.51	91.35	54.15	91.08	66.40	83.60	50.98	85.40	55.55	87.25
	200	50.45	84.04	55.51	90.93	54.77	91.08	65.80	82.60	52.36	85.50	55.78	86.83
HLBL	50	53.15	85.07	59.97	89.82	56.41	90.95	56.60	83.40	62.38	85.66	57.70	86.98
	100	53.80	84.68	61.09	91.21	56.91	90.95	67.60	84.00	67.88	85.64	61.46	87.30
HPCA	50	50.45	85.20	55.51	91.35	52.39	89.95	62.20	83.00	51.02	83.50	54.31	86.60
	100	50.45	85.20	55.51	91.07	49.87	90.08	66.20	83.60	50.51	84.86	54.51	86.96
	200	50.45	85.20	55.51	90.10	49.87	89.82	68.00	82.80	50.38	85.72	54.84	86.73
GV6B	50	50.97	85.07	64.16	91.49	55.28	91.08	57.00	85.00	69.48	85.48	59.38	87.62
	100	50.58	84.94	60.53	89.68	63.82	91.08	58.00	84.60	70.22	85.51	60.63	87.16
	200	51.22	85.20	64.99	91.49	85.05	90.08	58.00	84.80	73.64	85.59	66.58	87.43
	300	56.24	85.33	70.15	89.68	89.07	91.21	72.40	85.20	75.48	85.79	72.67	87.44
GV840B	300	66.02	84.81	77.96	89.68	89.82	91.08	77.80	86.40	77.57	85.73	77.83	87.54
SkGr	300	67.95	82.50	81.59	89.40	95.10	94.60	80.80	88.40	80.93	85.92	81.27	88.16
one-hot		84.56		89.96		90.58		85.80		87.40		87.66	

Table 4: Accuracy results for RPA against standard PA both run for a single iteration.

rates. As for dimensions, embeddings of length 100 and more perform the best, although they involve estimating larger number of parameters, which is a priori difficult given the small sizes of the datasets.

By comparison with previous work on *imdb*, the RPA performance are substantially better than those reported by (Labutov and Lipson, 2013), whose best re-embedding score is 81.15 with CnW. By comparison, our best score with CnW is 86.64, and 88.52 with SkGr, thus closing the gap on (Maas et al., 2011) who report an accuracy of 88.89 using a much more computationally intensive approach specifically tailored to sentiment analysis. Interestingly, (Labutov and Lipson, 2013) show that accuracy can be further improved by concatenating re-embedded and 1-hot representations. This option is also available to us, but we leave it to future work.

Finally, Table 4 report accuracy results for RPA against PA when both algorithms are trained in a genuine online mode, that is with a single pass over the data. As expected, performance drop for the RPA and the PA, but the decreases are comparatively much smaller for the RPA (from 2% to 3%) compared to the PA (from 0.4% to 14%).

6 Conclusion and Future Work

In this paper, we have proposed a new scalable algorithm for learning word representations that are specifically tailored to a classification objective. This algorithm generalizes the well-known Passive-Aggressive algorithm, and we showed how to extend the regret bounds results of the PA to the RPA when either the weight vector or the embedding matrix is fixed. In addition, we have also provided synthetic and NLP experiments, demonstrating that the good classification performance of RPA.

In future work, we first would like to achieve a more complete analysis of the RPA algorithm when both $\boldsymbol{w}$ and $\boldsymbol{\Phi}$ both get updated. Also, we intend to investigate potential exact methods for solving biconvex minimization (Floudas and Viswewaran, 1990), as well as to develop a stochastic version of RPA, thus foregoing running the inner alternate search to convergence. More empirical perspectives include extending the RPA to linguistic structured prediction tasks, better handling of unknown words, and a deeper intrinsic and statistical evaluation of the embeddings learned by the RPA.

Acknowledgments

We thank the three anonymous reviewers for their comments. Pascal Denis was supported by ANR Grant GRASP No. ANR-16-CE33-0011, as well as by a grant from CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020.

References

Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2014. Tailoring continuous word representations for dependency parsing. In *ACL (2)*, pages 809–815.

Mathieu Blondel, Yotaro Kubo, and Ueda Naonori. 2014. Online passive-aggressive algorithms for non-negative matrix factorization and completion. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pages 96–104.

Yanqing Chen, Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2013. The expressive power of word embeddings. *arXiv preprint arXiv:1301.3226*.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585.

C. A. Floudas and V. Viswewaran. 1990. A global optimization algorithm (gop) for certain classes of nonconvex npls. i, theory. *Computers & chemical engineering*, 14(12):1397–1417.

Jochen Gorski, Frank Pfeuffer, and Kathrin Klamroth. 2007. Biconvex sets and optimization with biconvex functions: a survey and extensions. *Mathematical Methods of Operations Research*, 66(3):373–407.

Y. Grandvalet and S. Canu. 2003. Adaptive scaling for feature selection in svms. In *Advances in Neural Information Processing Systems*, volume 15. MIT Press.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*.

Igor Labutov and Hod Lipson. 2013. Re-embedding words. In *Proceedings of the 51th Annual Meeting of the Association for Computational Linguistics*.

Rémi Lebret and Ronan Collobert. 2014. Word emdeddings through hellinger PCA. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*.

Rémi Lebret, Joël Legrand, and Ronan Collobert. 2013. Is deep learning really necessary for word embeddings? In *NIPS Workshop on Deep Learning*.

Omer Levy and Yoav Goldberg. 2014a. Dependency-based word embeddings. In *ACL (2)*, pages 302–308.

Omer Levy and Yoav Goldberg. 2014b. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Andriy Mnih and Geoffrey E Hinton. 2009. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pages 1081–1088.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–43.

Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. 2015. Evaluation methods for unsupervised word embeddings. In *Proc. of EMNLP*.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics.

Mengqiu Wang and Christopher D Manning. 2013. Effect of non-linear deep architecture in sequence labeling. In *IJCNLP*, pages 1285–1291.