Distributed Training Strategies for the Structured Perceptron

Ryan McDonald Keith Hall Gideon Mann
Google, Inc., New York / Zurich
{ryanmcd|kbhall|gmann}@google.com

Abstract

Perceptron training is widely applied in the natural language processing community for learning complex structured models. Like all structured prediction learning frameworks, the structured perceptron can be costly to train as training complexity is proportional to inference, which is frequently non-linear in example sequence length. In this paper we investigate distributed training strategies for the structured perceptron as a means to reduce training times when computing clusters are available. We look at two strategies and provide convergence bounds for a particular mode of distributed structured perceptron training based on iterative parameter mixing (or averaging). We present experiments on two structured prediction problems – named-entity recognition and dependency parsing – to highlight the efficiency of this method.

1 Introduction

One of the most popular training algorithms for structured prediction problems in natural language processing is the perceptron (Rosenblatt, 1958; Collins, 2002). The structured perceptron has many desirable properties, most notably that there is no need to calculate a partition function, which is necessary for other structured prediction paradigms such as CRFs (Lafferty et al., 2001). Furthermore, it is robust to approximate inference, which is often required for problems where the search space is too large and where strong structural independence assumptions are insufficient, such as parsing (Collins and Roark, 2004; McDonald and Pereira, 2006; Zhang and Clark, 2008) and machine translation (Liang et al., 2006). However, like all structured prediction learning frameworks, the structure perceptron can still be cumbersome to train. This is both due to the increasing size of available training sets as well as the fact that training complexity is proportional to inference, which is frequently non-linear in sequence length, even with strong structural independence assumptions.

In this paper we investigate distributed training strategies for the structured perceptron as a means of reducing training times when large computing clusters are available. Traditional machine learning algorithms are typically designed for a single machine, and designing an efficient training mechanism for analogous algorithms on a computing cluster – often via a map-reduce framework (Dean and Ghemawat, 2004) – is an active area of research (Chu et al., 2007). However, unlike many batch learning algorithms that can easily be distributed through the gradient calculation, a distributed training analog for the perceptron is less clear cut. It employs online updates and its loss function is technically non-convex.

A recent study by Mann et al. (2009) has shown that distributed training through parameter mixing (or averaging) for maximum entropy models can be empirically powerful and has strong theoretical guarantees. A parameter mixing strategy, which can be applied to any parameterized learning algorithm, trains separate models in parallel, each on a disjoint subset of the training data, and then takes an average of all the parameters as the final model. In this paper, we provide results which suggest that the perceptron is ill-suited for straight-forward parameter mixing, even though it is commonly used for large-scale structured learning, e.g., Whitelaw et al. (2008) for named-entity recognition. However, a slight mod-

ification we call *iterative parameter mixing* can be shown to: 1) have similar convergence properties to the standard perceptron algorithm, 2) find a separating hyperplane if the training set is separable, 3) reduce training times significantly, and 4) produce models with comparable (or superior) accuracies to those trained serially on all the data.

2 Related Work

Distributed cluster computation for many batch training algorithms has previously been examined by Chu et al. (2007), among others. Much of the relevant prior work on online (or sub-gradient) distributed training has been focused on asynchronous optimization via gradient descent. In this scenario, multiple machines run stochastic gradient descent simultaneously as they update and read from a shared parameter vector asynchronously. Early work by Tsitsiklis et al. (1986) demonstrated that if the delay between model updates and reads is bounded, then asynchronous optimization is guaranteed to converge. Recently, Zinkevich et al. (2009) performed a similar type of analysis for online learners with asynchronous updates via stochastic gradient descent. The asynchronous algorithms in these studies require shared memory between the distributed computations and are less suitable to the more common cluster computing environment, which is what we study here.

While we focus on the perceptron algorithm, there is a large body of work on training structured prediction classifiers. For batch training the most common is conditional random fields (CRFs) (Lafferty et al., 2001), which is the structured analog of maximum entropy. As such, its training can easily be distributed through the gradient or sub-gradient computations (Finkel et al., 2008). However, unlike perceptron, CRFs require the computation of a partition function, which is often expensive and sometimes intractable. Other batch learning algorithms include M^{3}Ns (Taskar et al., 2004) and Structured SVMs (Tsochantaridis et al., 2004). Due to their efficiency, online learning algorithms have gained attention, especially for structured prediction tasks in NLP. In addition to the perceptron (Collins, 2002), others have looked at stochastic gradient descent (Zhang, 2004), passive aggressive algorithms (McDonald et al., 2005; Crammer et al., 2006), the recently introduced confidence weighted learning (Dredze et al., 2008) and coordinate descent algorithms (Duchi and Singer, 2009).

Perceptron($\mathcal{T} = \{(\mathbf{x}_t, \mathbf{y}_t)\}_{t=1}^{|\mathcal{T}|}$)
1. $\mathbf{w}^{(0)} = \mathbf{0}$; $k = 0$
2. for $n : 1..N$
3. for $t : 1..T$
4. Let $\mathbf{y}' = \arg\max_{\mathbf{y}'} \mathbf{w}^{(k)} \cdot \mathbf{f}(\mathbf{x}_t, \mathbf{y}')$
5. if $\mathbf{y}' \neq \mathbf{y}_t$
6. $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \mathbf{f}(\mathbf{x}_t, \mathbf{y}_t) - \mathbf{f}(\mathbf{x}_t, \mathbf{y}')$
7. $k = k + 1$
8. return $\mathbf{w}^{(k)}$

Figure 1: The perceptron algorithm.

3 Structured Perceptron

The structured perceptron was introduced by Collins (2002) and we adopt much of the notation and presentation of that study. The structured percetron algorithm – which is identical to the multi-class perceptron – is shown in Figure 1. The perceptron is an online learning algorithm and processes training instances one at a time during each epoch of training. Lines 4-6 are the core of the algorithm. For a input-output training instance pair $(\mathbf{x}_t, \mathbf{y}_t) \in \mathcal{T}$, the algorithm predicts a structured output $\mathbf{y}' \in \mathcal{Y}_t$, where $\mathcal{Y}_t$ is the space of permissible structured outputs for input $\mathbf{x}_t$, e.g., parse trees for an input sentence. This prediction is determined by a linear classifier based on the dot product between a high-dimensional feature representation of a candidate input-output pair $\mathbf{f}(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^M$ and a corresponding weight vector $\mathbf{w} \in \mathbb{R}^M$, which are the parameters of the model[1]. If this prediction is incorrect, then the parameters are updated to add weight to features for the corresponding correct output $\mathbf{y}_t$ and take weight away from features for the incorrect output $\mathbf{y}'$. For structured prediction, the inference step in line 4 is problem dependent, e.g., CKY for context-free parsing.

A training set $\mathcal{T}$ is separable with margin $\gamma > 0$ if there exists a vector $\mathbf{u} \in \mathbb{R}^M$ with $\|\mathbf{u}\| = 1$ such that $\mathbf{u} \cdot \mathbf{f}(\mathbf{x}_t, \mathbf{y}_t) - \mathbf{u} \cdot \mathbf{f}(\mathbf{x}_t, \mathbf{y}') \geq \gamma$, for all $(\mathbf{x}_t, \mathbf{y}_t) \in \mathcal{T}$, and for all $\mathbf{y}' \in \mathcal{Y}_t$ such that $\mathbf{y}' \neq \mathbf{y}_t$. Furthermore, let $R \geq ||\mathbf{f}(\mathbf{x}_t, \mathbf{y}_t) - \mathbf{f}(\mathbf{x}_t, \mathbf{y}')||$, for all $(\mathbf{x}_t, \mathbf{y}_t) \in \mathcal{T}$ and $\mathbf{y}' \in \mathcal{Y}_t$. A fundamental theorem

[1]The perceptron can be kernalized for non-linearity.

of the perceptron is as follows:

Theorem 1 (Novikoff (1962))**.** *Assume training set $\mathcal{T}$ is separable by margin γ. Let k be the number of mistakes made training the perceptron (Figure 1) on $\mathcal{T}$. If training is run indefinitely, then $k \leq \frac{R^2}{\gamma^2}$.*

Proof. See Collins (2002) Theorem 1. □

Theorem 1 implies that if $\mathcal{T}$ is separable then 1) the perceptron will converge in a finite amount of time, and 2) will produce a $\mathbf{w}$ that separates $\mathcal{T}$. Collins also proposed a variant of the structured perceptron where the final weight vector is a weighted average of all parameters that occur during training, which he called the *averaged perceptron* and can be viewed as an approximation to the voted perceptron algorithm (Freund and Schapire, 1999).

4 Distributed Structured Perceptron

In this section we examine two distributed training strategies for the perceptron algorithm based on parameter mixing.

4.1 Parameter Mixing

Distributed training through parameter mixing is a straight-forward way of training classifiers in parallel. The algorithm is given in Figure 2. The idea is simple: divide the training data $\mathcal{T}$ into S disjoint *shards* such that $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_S\}$. Next, train perceptron models (or any learning algorithm) on each shard in parallel. After training, set the final parameters to a weighted mixture of the parameters of each model using mixture coefficients $\boldsymbol{\mu}$. Note that we call this strategy *parameter mixing* as opposed to *parameter averaging* to distinguish it from the averaged perceptron (see previous section). It is easy to see how this can be implemented on a cluster through a map-reduce framework, i.e., the map step trains the individual models in parallel and the reduce step mixes their parameters. The advantages of parameter mixing are: 1) that it is parallel, making it possibly to scale to extremely large data sets, and 2) it is resource efficient, in particular with respect to network usage as parameters are not repeatedly passed across the network as is often the case for exact distributed training strategies.

For maximum entropy models, Mann et al. (2009) show it is possible to bound the norm of the difference between parameters trained on all the data serially versus parameters trained with parameter mixing. However, their analysis requires a stability bound on the parameters of a regularized maximum entropy model, which is not known to hold for the perceptron. In Section 5, we present empirical results showing that parameter mixing for distributed perceptron can be sub-optimal. Additionally, Dredze et al. (2008) present negative parameter mixing results for confidence weighted learning, which is another online learning algorithm. The following theorem may help explain this behavior.

PerceptronParamMix($\mathcal{T} = \{(\mathbf{x}_t, \mathbf{y}_t)\}_{t=1}^{|\mathcal{T}|}$)
1. Shard $\mathcal{T}$ into S pieces $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_S\}$
2. $\mathbf{w}^{(i)}$ = Perceptron($\mathcal{T}_i$) †
3. $\mathbf{w} = \sum_i \mu_i \mathbf{w}^{(i)}$ ‡
4. return $\mathbf{w}$

Figure 2: Distributed perceptron using a parameter mixing strategy. † Each $\mathbf{w}^{(i)}$ is computed in parallel. ‡ $\boldsymbol{\mu} = \{\mu_1, \ldots, \mu_S\}, \forall \mu_i \in \boldsymbol{\mu} : \mu_i \geq 0$ and $\sum_i \mu_i = 1$.

Theorem 2. *For a any training set $\mathcal{T}$ separable by margin γ, the perceptron algorithm trained through a parameter mixing strategy (Figure 2) does not necessarily return a separating weight vector* $\mathbf{w}$.

Proof. Consider a binary classification setting where $\mathcal{Y} = \{0, 1\}$ and $\mathcal{T}$ has 4 instances. We distribute the training set into two shards, $\mathcal{T}_1 = \{(\mathbf{x}_{1,1}, \mathbf{y}_{1,1}), (\mathbf{x}_{1,2}, \mathbf{y}_{1,2})\}$ and $\mathcal{T}_2 = \{(\mathbf{x}_{2,1}, \mathbf{y}_{2,1}), (\mathbf{x}_{2,2}, \mathbf{y}_{2,2})\}$. Let $\mathbf{y}_{1,1} = \mathbf{y}_{2,1} = 0$ and $\mathbf{y}_{1,2} = \mathbf{y}_{2,2} = 1$. Now, let $\mathbf{w}, \mathbf{f} \in \mathbb{R}^6$ and using block features, define the feature space as,

$\mathbf{f}(\mathbf{x}_{1,1}, 0) = [1\ 1\ 0\ 0\ 0\ 0]$ $\quad$ $\mathbf{f}(\mathbf{x}_{1,1}, 1) = [0\ 0\ 0\ 1\ 1\ 0]$
$\mathbf{f}(\mathbf{x}_{1,2}, 0) = [0\ 0\ 1\ 0\ 0\ 0]$ $\quad$ $\mathbf{f}(\mathbf{x}_{1,2}, 1) = [0\ 0\ 0\ 0\ 0\ 1]$
$\mathbf{f}(\mathbf{x}_{2,1}, 0) = [0\ 1\ 1\ 0\ 0\ 0]$ $\quad$ $\mathbf{f}(\mathbf{x}_{2,1}, 1) = [0\ 0\ 0\ 0\ 1\ 1]$
$\mathbf{f}(\mathbf{x}_{2,2}, 0) = [1\ 0\ 0\ 0\ 0\ 0]$ $\quad$ $\mathbf{f}(\mathbf{x}_{2,2}, 1) = [0\ 0\ 0\ 1\ 0\ 0]$

Assuming label 1 tie-breaking, parameter mixing returns $\mathbf{w}^1$=[1 1 0 -1 -1 0] and $\mathbf{w}^2$=[0 1 1 0 -1 -1]. For any $\boldsymbol{\mu}$, the mixed weight vector $\mathbf{w}$ will not separate all the points. If both μ_1/μ_2 are non-zero, then all examples will be classified 0. If μ_1=1 and μ_2=0, then $(\mathbf{x}_{2,2}, \mathbf{y}_{2,2})$ will be incorrectly classified as 0 and $(\mathbf{x}_{1,2}, \mathbf{y}_{1,2})$ when μ_1=0 and μ_2=1. But there is a separating weight vector $\mathbf{w}$ = [-1 2 -1 1 -2 1]. □

This counter example does not say that a parameter mixing strategy will not converge. On the contrary,

if $\mathcal{T}$ is separable, then each of its subsets is separable and converge via Theorem 1. What it does say is that, independent of $\boldsymbol{\mu}$, the mixed weight vector produced after convergence will not necessarily separate the entire data, even when $\mathcal{T}$ is separable.

4.2 Iterative Parameter Mixing

Consider a slight augmentation to the parameter mixing strategy. Previously, each parallel perceptron was trained to convergence before the parameter mixing step. Instead, shard the data as before, but train a single epoch of the perceptron algorithm for each shard (in parallel) and mix the model weights. This mixed weight vector is then re-sent to each shard and the perceptrons on those shards reset their weights to the new mixed weights. Another single epoch of training is then run (again in parallel over the shards) and the process repeats. This *iterative parameter mixing* algorithm is given in Figure 3.

Again, it is easy to see how this can be implemented as map-reduce, where the map computes the parameters for each shard for one epoch and the reduce mixes and re-sends them. This is analogous to batch distributed gradient descent methods where the gradient for each shard is computed in parallel in the map step and the reduce step sums the gradients and updates the weight vector. The disadvantage of iterative parameter mixing, relative to simple parameter mixing, is that the amount of information sent across the network will increase. Thus, if network latency is a bottleneck, this can become problematic. However, for many parallel computing frameworks, including both multi-core computing as well as cluster computing with high rates of connectivity, this is less of an issue.

Theorem 3. *Assume a training set $\mathcal{T}$ is separable by margin γ. Let $k_{i,n}$ be the number of mistakes that occurred on shard i during the nth epoch of training. For any N, when training the perceptron with iterative parameter mixing (Figure 3),*

$$\sum_{n=1}^{N}\sum_{i=1}^{S}\mu_{i,n}k_{i,n} \leq \frac{R^2}{\gamma^2}$$

Proof. Let $\mathbf{w}^{(i,n)}$ to be the weight vector for the ith shard after the nth epoch of the main loop and let $\mathbf{w}^{([i,n]-k)}$ be the weight vector that existed on shard i in the nth epoch k errors before $\mathbf{w}^{(i,n)}$. Let

PerceptronIterParamMix($\mathcal{T} = \{(\mathbf{x}_t, \mathbf{y}_t)\}_{t=1}^{|\mathcal{T}|}$)
1. Shard $\mathcal{T}$ into S pieces $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_S\}$
2. $\mathbf{w} = \mathbf{0}$
3. for $n : 1..N$
4. $\quad \mathbf{w}^{(i,n)} = \text{OneEpochPerceptron}(\mathcal{T}_i, \mathbf{w})$ †
5. $\quad \mathbf{w} = \sum_i \mu_{i,n}\mathbf{w}^{(i,n)}$ ‡
6. return $\mathbf{w}$

OneEpochPerceptron($\mathcal{T}$, $\mathbf{w}^*$)
1. $\mathbf{w}^{(0)} = \mathbf{w}^*;\ k = 0$
2. for $t : 1..T$
3. $\quad$ Let $\mathbf{y}' = \arg\max_{\mathbf{y}'} \mathbf{w}^{(k)} \cdot \mathbf{f}(\mathbf{x}_t, \mathbf{y}')$
4. $\quad$ if $\mathbf{y}' \neq \mathbf{y}_t$
5. $\quad\quad \mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \mathbf{f}(\mathbf{x}_t, \mathbf{y}_t) - \mathbf{f}(\mathbf{x}_t, \mathbf{y}')$
6. $\quad\quad k = k + 1$
7. return $\mathbf{w}^{(k)}$

Figure 3: Distributed perceptron using an iterative parameter mixing strategy. † Each $\mathbf{w}^{(i,n)}$ is computed in parallel. ‡ $\boldsymbol{\mu}_n = \{\mu_{1,n}, \ldots, \mu_{S,n}\}$, $\forall \mu_{i,n} \in \boldsymbol{\mu}_n$: $\mu_{i,n} \geq 0$ and $\forall n$: $\sum_i \mu_{i,n} = 1$.

$\mathbf{w}^{(\text{avg},n)}$ be the mixed vector from the weight vectors returned after the nth epoch, i.e.,

$$\mathbf{w}^{(\text{avg},n)} = \sum_{i=1}^{S}\mu_{i,n}\mathbf{w}^{(i,n)}$$

Following the analysis from Collins (2002) Theorem 1, by examining line 5 of OneEpochPerceptron in Figure 3 and the fact that $\mathbf{u}$ separates the data by γ:

$$\begin{aligned}
\mathbf{u}\cdot\mathbf{w}^{(i,n)} &= \mathbf{u}\cdot\mathbf{w}^{([i,n]-1)} \\
&\quad + \mathbf{u}\cdot(\mathbf{f}(\mathbf{x}_t,\mathbf{y}_t) - \mathbf{f}(\mathbf{x}_t,\mathbf{y}')) \\
&\geq \mathbf{u}\cdot\mathbf{w}^{([i,n]-1)} + \gamma \\
&\geq \mathbf{u}\cdot\mathbf{w}^{([i,n]-2)} + 2\gamma \\
\ldots &\geq \mathbf{u}\cdot\mathbf{w}^{(\text{avg},n-1)} + k_{i,n}\gamma \quad \text{(A1)}
\end{aligned}$$

That is, $\mathbf{u}\cdot\mathbf{w}^{(i,n)}$ is bounded below by the average weight vector for the n-1st epoch plus the number of mistakes made on shard i during the nth epoch times the margin γ. Next, by OneEpochPerceptron line 5, the definition of R, and $\mathbf{w}^{([i,n]-1)}(\mathbf{f}(\mathbf{x}_t,\mathbf{y}_t) - \mathbf{f}(\mathbf{x}_t,\mathbf{y}')) \leq 0$ when line 5 is called:

$$\begin{aligned}
\|\mathbf{w}^{(i,n)}\|^2 &= \|\mathbf{w}^{([i,n]-1)}\|^2 \\
&\quad + \|\mathbf{f}(\mathbf{x}_t,\mathbf{y}_t) - \mathbf{f}(\mathbf{x}_t,\mathbf{y}')\|^2 \\
&\quad + 2\mathbf{w}^{([i,n]-1)}(\mathbf{f}(\mathbf{x}_t,\mathbf{y}_t) - \mathbf{f}(\mathbf{x}_t,\mathbf{y}')) \\
&\leq \|\mathbf{w}^{([i,n]-1)}\|^2 + R^2 \\
&\leq \|\mathbf{w}^{([i,n]-2)}\|^2 + 2R^2 \\
\ldots &\leq \|\mathbf{w}^{(\text{avg},n-1)}\|^2 + k_{i,n}R^2 \quad \text{(A2)}
\end{aligned}$$

That is, the squared L2-norm of a shards weight vector is bounded above by the same value for the average weight vector of the n-1st epoch and the number of mistakes made on that shard during the nth epoch times R^2.

Using A1/A2 we prove two inductive hypotheses:

$$\mathbf{u} \cdot \mathbf{w}^{(\text{avg},N)} \geq \sum_{n=1}^{N} \sum_{i=1}^{S} \mu_{i,n} k_{i,n} \gamma \quad \text{(IH1)}$$

$$\|\mathbf{w}^{(\text{avg},N)}\|^2 \leq \sum_{n=1}^{N} \sum_{i=1}^{S} \mu_{i,n} k_{i,n} R^2 \quad \text{(IH2)}$$

IH1 implies $\|\mathbf{w}^{(\text{avg},N)}\| \geq \sum_{n=1}^{N} \sum_{i=1}^{S} \mu_{i,n} k_{i,n} \gamma$ since $\mathbf{u} \cdot \mathbf{w} \leq \|\mathbf{u}\| \|\mathbf{w}\|$ and $\|\mathbf{u}\| = 1$.

The base case is $\mathbf{w}^{(\text{avg},1)}$, where we can observe:

$$\mathbf{u} \cdot \mathbf{w}^{\text{avg},1} = \sum_{i=1}^{S} \mu_{i,1} \mathbf{u} \cdot \mathbf{w}^{(i,1)} \geq \sum_{i=1}^{S} \mu_{i,1} k_{i,1} \gamma$$

using A1 and the fact that $\mathbf{w}^{(\text{avg},0)} = \mathbf{0}$ for the second step. For the IH2 base case we can write:

$$\begin{aligned} \|\mathbf{w}^{(\text{avg},1)}\|^2 &= \left\| \sum_{i=1}^{S} \mu_{i,1} \mathbf{w}^{(i,1)} \right\|^2 \\ &\leq \sum_{i=1}^{S} \mu_{i,1} \|\mathbf{w}^{(i,1)}\|^2 \leq \sum_{i=1}^{S} \mu_{i,1} k_{i,1} R^2 \end{aligned}$$

The first inequality is Jensen's inequality, and the second is true by A2 and $\|\mathbf{w}^{(\text{avg},0)}\|^2 = 0$.

Proceeding to the general case, $\mathbf{w}^{(\text{avg},N)}$:

$$\begin{aligned} \mathbf{u} \cdot \mathbf{w}^{(\text{avg},N)} &= \sum_{i=1}^{S} \mu_{i,N} (\mathbf{u} \cdot \mathbf{w}^{(i,N)}) \\ &\geq \sum_{i=1}^{S} \mu_{i,N} (\mathbf{u} \cdot \mathbf{w}^{(\text{avg},N-1)} + k_{i,N} \gamma) \\ &= \mathbf{u} \cdot \mathbf{w}^{(\text{avg},N-1)} + \sum_{i=1}^{S} \mu_{i,N} k_{i,N} \gamma \\ &\geq \left[\sum_{n=1}^{N-1} \sum_{i=1}^{S} \mu_{i,n} k_{i,n} \gamma \right] + \sum_{i=1}^{S} \mu_{i,N} k_{i,N} \\ &= \sum_{n=1}^{N} \sum_{i=1}^{S} \mu_{i,n} k_{i,n} \gamma \end{aligned}$$

The first inequality uses A1, the second step $\sum_i \mu_{i,N} = 1$ and the second inequality the inductive hypothesis IH1. For IH2, in the general case, we can write:

$$\begin{aligned} \|\mathbf{w}^{(\text{avg},N)}\|^2 &\leq \sum_{i=1}^{S} \mu_{i,N} \|\mathbf{w}^{(i,N)}\|^2 \\ &\leq \sum_{i=1}^{S} \mu_{i,N} (\|\mathbf{w}^{(\text{avg},N-1)}\|^2 + k_{i,N} R^2) \\ &= \|\mathbf{w}^{(\text{avg},N-1)}\|^2 + \sum_{i=1}^{S} \mu_{i,N} k_{i,N} R^2 \\ &\leq \left[\sum_{n=1}^{N-1} \sum_{i=1}^{S} \mu_{i,n} k_{i,n} R^2 \right] + \sum_{i=1}^{S} \mu_{i,N} k_{i,N} R^2 \\ &= \sum_{n=1}^{N} \sum_{i=1}^{S} \mu_{i,n} k_{i,n} R^2 \end{aligned}$$

The first inequality is Jensen's, the second A2, and the third the inductive hypothesis IH2. Putting together IH1, IH2 and $\|\mathbf{w}^{(\text{avg},N)}\| \geq \mathbf{u} \cdot \mathbf{w}^{(\text{avg},N)}$:

$$\left[\sum_{n=1}^{N} \sum_{i=1}^{S} \mu_{i,n} k_{i,n} \right]^2 \gamma^2 \leq \left[\sum_{n=1}^{N} \sum_{i=1}^{S} \mu_{i,n} k_{i,n} \right] R^2$$

which yields: $\sum_{n=1}^{N} \sum_{i=1}^{S} \mu_{i,n} k_{i,n} \leq \frac{R^2}{\gamma^2}$ □

4.3 Analysis

If we set each $\boldsymbol{\mu}_n$ to be the uniform mixture, $\mu_{i,n} = 1/S$, then *Theorem 3 guarantees convergence to a separating hyperplane*. If $\sum_{i=1}^{S} \mu_{i,n} k_{i,n} = 0$, then the previous weight vector already separated the data. Otherwise, $\sum_{n=1}^{N} \sum_{i=1}^{S} \mu_{i,n} k_{i,n}$ is still increasing, but is bounded and cannot increase indefinitely. Also note that if $S = 1$, then $\mu_{1,n}$ must equal 1 for all n and this bound is identical to Theorem 1.

However, we are mainly concerned with how fast convergence occurs, which is directly related to the number of training epochs each algorithm must run, i.e., N in Figure 1 and Figure 3. For the non-distributed variant of the perceptron we can say that $N_{\text{non\_dist}} \leq R^2/\gamma^2$ since in the worst case a single mistake happens on each epoch.[2] For the distributed case, consider setting $\mu_{i,n} = k_{i,n}/k_n$, where $k_n = \sum_i k_{i,n}$. That is, we mix parameters proportional to the number of errors each made during the previous epoch. Theorem 3 still implies convergence to a separating hyperplane with this choice. Further, we can

[2] It is not hard to derive such degenerate cases.

bound the required number of epochs N_{dist}:

$$N_{\text{dist}} \leq \sum_{n=1}^{N_{\text{dist}}} \prod_{i=1}^{S} [k_{i,n}]^{\frac{k_{i,n}}{k_n}} \leq \sum_{n=1}^{N_{\text{dist}}} \sum_{i=1}^{S} \frac{k_{i,n}}{k_n} k_{i,n} \leq \frac{R^2}{\gamma^2}$$

Ignoring when all $k_{i,n}$ are zero (since the algorithm will have converged), the first inequality is true since either $k_{i,n} \geq 1$, implying that $[k_{i,n}]^{k_{i,n}/k_n} \geq 1$, or $k_{i,n} = 0$ and $[k_{i,n}]^{k_{i,n}/k_n} = 1$. The second inequality is true by the generalized arithmetic-geometric mean inequality and the final inequality is Theorem 3. Thus, the worst-case number of epochs is identical for both the regular and distributed perceptron – but the distributed perceptron can theoretically process each epoch S times faster. This observation holds only for cases where $\mu_{i,n} > 0$ when $k_{i,n} \geq 1$ and $\mu_{i,n} = 0$ when $k_{i,n} = 0$, which does not include uniform mixing.

5 Experiments

To investigate the distributed perceptron strategies discussed in Section 4 we look at two structured prediction tasks – named entity recognition and dependency parsing. We compare up to four systems:

1. **Serial (All Data)**: This is the classifier returned if trained serially on all the available data.
2. **Serial (Sub Sampling)**: Shard the data, select one shard randomly and train serially.
3. **Parallel (Parameter Mix)**: Parallel strategy discussed in Section 4.1 with uniform mixing.
4. **Parallel (Iterative Parameter Mix)**: Parallel strategy discussed in Section 4.2 with uniform mixing (Section 5.1 looks at mixing strategies).

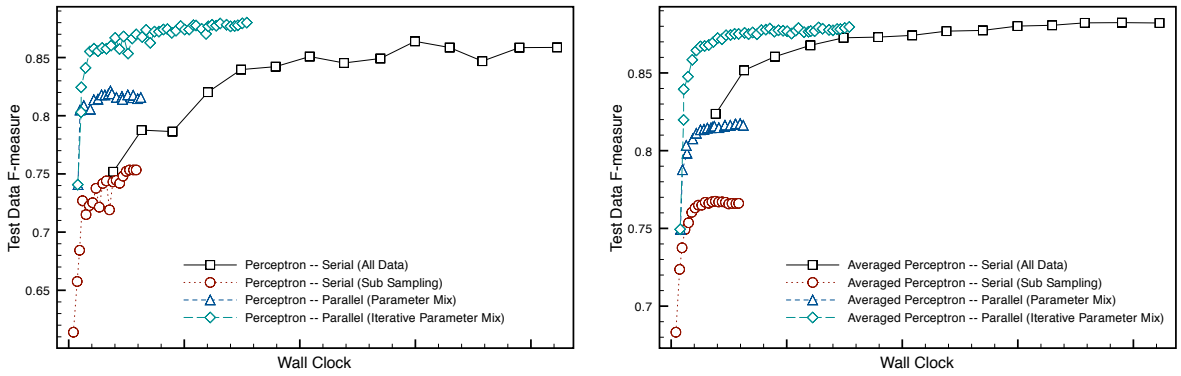
For all four systems we compare results for both the standard perceptron algorithm as well as the averaged perceptron algorithm (Collins, 2002).

We report the final test set metrics of the converged classifiers to determine whether any loss in accuracy is observed as a consequence of distributed training strategies. We define convergence as either: 1) the training set is separated, or 2) the training set performance measure (accuracy, f-measure, etc.) does not change by more than some pre-defined threshold on three consecutive epochs. As with most real world data sets, convergence by training set separation was rarely observed, though in both cases training set accuracies approached 100%. For both tasks we also plot test set metrics relative to the user wall-clock taken to obtain the classifier. The results were computed by collecting the metrics at the end of each epoch for every classifier. All experiments used 10 shards (Section 5.1 looks at convergence relative to different shard size).

Our first experiment is a named-entity recognition task using the English data from the CoNLL 2003 shared-task (Tjong Kim Sang and De Meulder, 2003). The task is to detect entities in sentences and label them as one of four types: people, organizations, locations or miscellaneous. For our experiments we used the entire training set (14041 sentences) and evaluated on the official development set (3250 sentences). We used a straight-forward IOB label encoding with a 1st order Markov factorization. Our feature set consisted of predicates extracted over word identities, word affixes, orthography, part-of-speech tags and corresponding concatenations. The evaluation metric used was micro f-measure over the four entity class types.

Results are given in Figure 4. There are a number of things to observe here: 1) training on a single shard clearly provides inferior performance to training on all data, 2) the simple parameter mixing strategy improves upon a single shard, but does not meet the performance of training on all data, 3) iterative parameter mixing achieves performance as good as or better than training serially on all the data, and 4) the distributed algorithms return better classifiers much quicker than training serially on all the data. This is true regardless of whether the underlying algorithm is the regular or the averaged perceptron. Point 3 deserves more discussion. In particular, the iterative parameter mixing strategy has a higher final f-measure than training on all the data serially than the standard perceptron (f-measure of 87.9 vs. 85.8). We suspect this happens for two reasons. First, the parameter mixing has a bagging like effect which helps to reduce the variance of the per-shard classifiers (Breiman, 1996). Second, the fact that parameter mixing is just a form of parameter averaging perhaps has the same effect as the averaged perceptron.

Our second set of experiments looked at the much more computationally intensive task of dependency parsing. We used the Prague Dependency Treebank (PDT) (Hajič et al., 2001), which is a Czech



	Reg. Perceptron F-measure	Avg. Perceptron F-measure
Serial (All Data)	85.8	88.2
Serial (Sub Sampling)	75.3	76.6
Parallel (Parameter Mix)	81.5	81.6
Parallel (Iterative Parameter Mix)	87.9	88.1

Figure 4: NER experiments. Upper figures plot test data f-measure versus wall clock for both regular perceptron (left) and averaged perceptron (right). Lower table is f-measure for converged models.

language treebank and currently one of the largest dependency treebanks in existence. We used the CoNLL-X training (72703 sentences) and testing splits (365 sentences) of this data (Buchholz and Marsi, 2006) and dependency parsing models based on McDonald and Pereira (2006) which factors features over pairs of dependency arcs in a tree. To parse all the sentences in the PDT, one must use a non-projective parsing algorithm, which is a known NP-complete inference problem when not assuming strong independence assumptions. Thus, the use of approximate inference techniques is common in order to find the highest weighted tree for a sentence. We use the approximate parsing algorithm given in McDonald and Pereira (2006), which runs in time roughly cubic in sentence length. To train such a model is computationally expensive and can take on the order of days to train on a single machine.

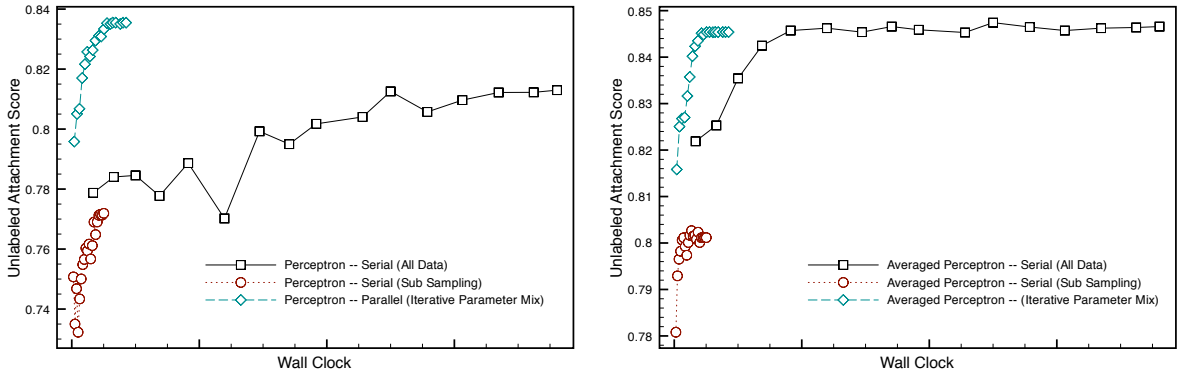
Unlabeled attachment scores (Buchholz and Marsi, 2006) are given in Figure 5. The same trends are seen for dependency parsing that are seen for named-entity recognition. That is, iterative parameter mixing learns classifiers faster and has a final accuracy as good as or better than training serially on all data. Again we see that the iterative parameter mixing model returns a more accurate classifier than the regular perceptron, but at about the same level as the averaged perceptron.

5.1 Convergence Properties

Section 4.3 suggests that different weighting strategies can lead to different convergence properties, in particular with respect to the number of epochs. For the named-entity recognition task we ran four experiments comparing two different mixing strategies – uniform mixing ($\mu_{i,n}$=$1/S$) and error mixing ($\mu_{i,n}$=$k_{i,n}/k_n$) – each with two shard sizes – $S = 10$ and $S = 100$. Figure 6 plots the number of training errors per epoch for each strategy.

We can make a couple observations. First, the mixing strategy makes little difference. The reason being that the number of observed errors per epoch is roughly uniform across shards, making both strategies ultimately equivalent. The other observation is that increasing the number of shards can slow down convergence when viewed relative to epochs[3]. Again, this appears in contradiction to the analysis in Section 4.3, which, at least for the case of error weighted mixtures, implied that the number of epochs to convergence was independent of the number of shards. But that analysis was based on worst-case scenarios where a single error occurs on a single shard at each epoch, which is unlikely to occur in real world data. Instead, consider the uni-

[3] As opposed to raw wall-clock/CPU time, which benefits from faster epochs the more shards there are.



	Reg. Perceptron Unlabeled Attachment Score	Avg. Perceptron Unlabeled Attachment Score
Serial (All Data)	81.3	84.7
Serial (Sub Sampling)	77.2	80.1
Parallel (Iterative Parameter Mix)	83.5	84.5

Figure 5: Dependency Parsing experiments. Upper figures plot test data unlabeled attachment score versus wall clock for both regular perceptron (left) and averaged perceptron (right). Lower table is unlabeled attachment score for converged models.

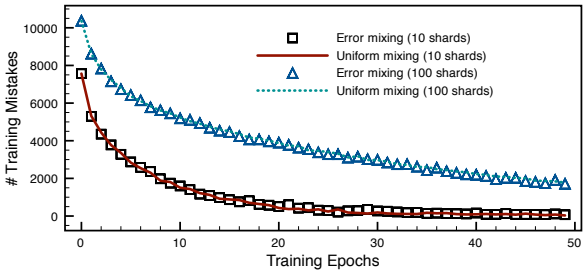


Figure 6: Training errors per epoch for different shard size and parameter mixing strategies.

form mixture case. Theorem 3 implies:

$$\sum_{n=1}^{N}\sum_{i=1}^{S}\frac{k_{i,n}}{S} \leq \frac{R^2}{\gamma^2} \quad \Longrightarrow \quad \sum_{n=1}^{N}\sum_{i=1}^{S} k_{i,n} \leq S \times \frac{R^2}{\gamma^2}$$

Thus, for cases where training errors are uniformly distributed across shards, it is possible that, in the worst-case, convergence may slow proportional the the number of shards. This implies a trade-off between slower convergence and quicker epochs when selecting a large number of shards. In fact, we observed a tipping point for our experiments in which increasing the number of shards began to have an adverse effect on training times, which for the named-entity experiments occurred around 25-50 shards. This is both due to reasons described in this section as well as the added overhead of maintaining and summing multiple high-dimensional weight vectors after each distributed epoch.

It is worth pointing out that a linear term S in the convergence bound above is similar to convergence/regret bounds for asynchronous distributed online learning, which typically have bounds linear in the asynchronous delay (Mesterharm, 2005; Zinkevich et al., 2009). This delay will be on average roughly equal to the number of shards S.

6 Conclusions

In this paper we have investigated distributing the structured perceptron via simple parameter mixing strategies. Our analysis shows that an iterative parameter mixing strategy is both guaranteed to separate the data (if possible) and significantly reduces the time required to train high accuracy classifiers. However, there is a trade-off between increasing training times through distributed computation and slower convergence relative to the number of shards. Finally, we note that using similar proofs to those given in this paper, it is possible to provide theoretical guarantees for distributed online passive aggressive learning (Crammer et al., 2006), which is a form of large-margin perceptron learning. Unfortunately space limitations prevent exploration here.

Acknowledgements: We thank Mehryar Mohri, Fernando Periera, Mark Dredze and the three anonymous reviews for their helpful comments on this work.

References

L. Breiman. 1996. Bagging predictors. *Machine Learning*, 24(2):123–140.

S. Buchholz and E. Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Conference on Computational Natural Language Learning*.

C.T. Chu, S.K. Kim, Y.A. Lin, Y.Y. Yu, G. Bradski, A.Y. Ng, and K. Olukotun. 2007. Map-Reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems*.

M. Collins and B. Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the Conference of the Association for Computational Linguistics*.

M. Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithm. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. 2006. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585.

J. Dean and S. Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. In *Sixth Symposium on Operating System Design and Implementation*.

M. Dredze, K. Crammer, and F. Pereira. 2008. Confidence-weighted linear classification. In *Proceedings of the International Conference on Machine learning*.

J. Duchi and Y. Singer. 2009. Efficient learning using forward-backward splitting. In *Advances in Neural Information Processing Systems*.

J.R. Finkel, A. Kleeman, and C.D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of the Conference of the Association for Computational Linguistics*.

Y. Freund and R.E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

J. Hajič, B. Vidova Hladka, J. Panevová, E. Hajičová, P. Sgall, and P. Pajas. 2001. Prague Dependency Treebank 1.0. LDC, 2001T10.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*.

P. Liang, A. Bouchard-Côté, D. Klein, and B. Taskar. 2006. An end-to-end discriminative approach to machine translation. In *Proceedings of the Conference of the Association for Computational Linguistics*.

G. Mann, R. McDonald, M. Mohri, N. Silberman, and D. Walker. 2009. Efficient large-scale distributed training of conditional maximum entropy models. In *Advances in Neural Information Processing Systems*.

R. McDonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics*.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the Conference of the Association for Computational Linguistics*.

C. Mesterharm. 2005. Online learning with delayed label feedback. In *Proceedings of Algorithmic Learning Theory*.

A.B. Novikoff. 1962. On convergence proofs on perceptrons. In *Symposium on the Mathematical Theory of Automata*.

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

B. Taskar, C. Guestrin, and D. Koller. 2004. Max-margin Markov networks. In *Advances in Neural Information Processing Systems*.

E. F. Tjong Kim Sang and F. De Meulder. 2003. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In *Proceedings of the Conference on Computational Natural Language Learning*.

J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans. 1986. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, 31(9):803–812.

I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the International Conference on Machine learning*.

C. Whitelaw, A. Kehlenbeck, N. Petrovic, and L. Ungar. 2008. Web-scale named entity recognition. In *Proceedings of the International Conference on Information and Knowledge Management*.

Y. Zhang and S. Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

T. Zhang. 2004. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the International Conference on Machine Learning*.

M. Zinkevich, A. Smola, and J. Langford. 2009. Slow learners are fast. In *Advances in Neural Information Processing Systems*.