Asynchronous Parallel Learning for Neural Networks and Structured Models with Dense Features

Xu Sun
∗MOE Key Laboratory of Computational Linguistics, Peking University
†School of Electronics Engineering and Computer Science, Peking University
xusun@pku.edu.cn

Abstract

Existing asynchronous parallel learning methods are only for the sparse feature models, and they face new challenges for the dense feature models like neural networks (e.g., LSTM, RNN). The problem for dense features is that asynchronous parallel learning brings gradient errors derived from overwrite actions. We show that gradient errors are very common and inevitable. Nevertheless, our theoretical analysis shows that the learning process with gradient errors can still be convergent towards the optimum of objective functions for many practical applications. Thus, we propose a simple method *AsynGrad* for asynchronous parallel learning with gradient error. Base on various dense feature models (LSTM, dense-CRF) and various NLP tasks, experiments show that *AsynGrad* achieves substantial improvement on training speed, and without any loss on accuracy.

1 Introduction

Stochastic learning methods can accelerate the training speed compared with traditional batch training methods. A widely used stochastic learning method is the stochastic gradient descent method (SGD) (Bertsekas, 1999; Bottou and Bousquet, 2008; Shalev-Shwartz and Srebro, 2008; Sun et al., 2012; Sun et al., 2014). For large-scale datasets, the SGD training methods can be much faster than batch training methods. For further improve the training speed over multi-core machines and clusters, a variety of asynchronous (lock-free) parallel learning methods has been developed based on stochastic learning (Niu et al., 2011; Mcmahan and Streeter, 2014). Those asynchronous methods have shown to be more efficient than the synchronous (locked) parallel learning versions (Langford et al., 2009; Gimpel et al., 2010). Other related work on parallel stochastic learning also includes (Zinkevich et al., 2010; Dekel et al., 2012; Recht and Re, 2013; Dean et al., 2012).

Existing asynchronous parallel learning methods are mainly for the sparse feature models, and feature sparseness is a major assumption for those parallel learning methods (Niu et al., 2011; Mcmahan and Streeter, 2014). For example, Niu et al. (2011) proposed an interesting asynchronous parallel learning method *HogWild* for strict sparse machine learning problems with sparse separable cost functions (e.g., sparse SVM, low-rank matrix completion). Niu et al. (2011) stated that the key idea that underlies their lock-free approach is that the targeted machine learning problems are sparse. More recently, Mcmahan and Streeter (2014) proposed a delay-tolerant asynchronous parallel learning method, which is an extension of the method of Niu et al. (2011). This asynchronous parallel learning method proposed by Mcmahan and Streeter (2014) also strictly requires the sparseness of the features.

The situation is different for the dense feature models like neural networks (e.g., LSTM, RNN). Since the existing asynchronous parallel learning methods are strictly for sparse feature models, it is no longer reasonable to apply those methods for the dense feature models like neural networks. To our knowledge, there is very limited study on developing asynchronous parallel learning methods for neural networks. In most cases only synchronous versions of parallel learning are applied to neural networks, such as

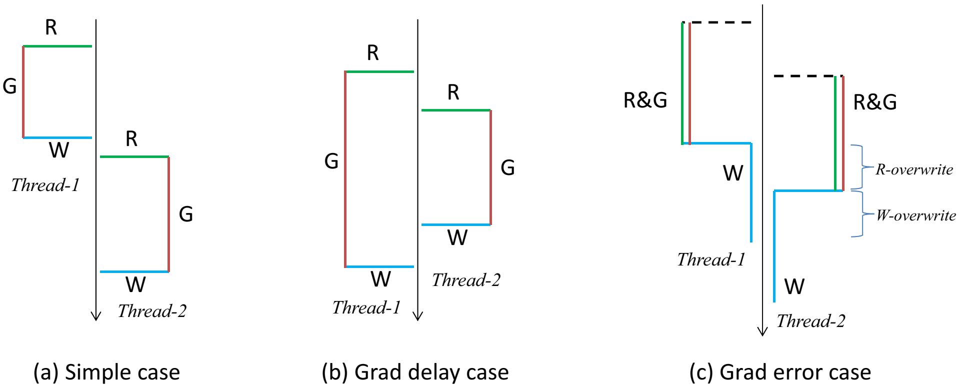

Figure 1: Illustrations of the *simple case* (left), the *gradient delay case* (middle), and the *gradient error case* (right) based on stochastic parallel learning. G means the gradient-computing action. R means the read action of shared memory. W means the write action.

GPU-based and mini-batch-based parallel learning methods. For GPU-based parallel learning, a matrix is computed in a synchronously parallelized way by using GPU units. For mini-batch-based parallel learning, the gradient of a mini-batch of samples are calculated in a synchronously parallelized way as well.

The major problem for applying asynchronous parallel learning to dense feature models is from the gradient errors. We show that gradient errors are very common and inevitable in applying asynchronous parallel learning to dense feature models. Suppose a dense feature model with 10 features/parameters, and we apply asynchronous (lock-free) parallel learning. When one thread is computing gradient, it needs to read the parameters from the shared memory. It is possible that 5 parameters are overwritten by another thread, which makes the computed gradient wrong. Not only there can be read-overwrite errors, but also there can be write-overwrite errors, as shown in Figure 1 (right).

Figure 1 illustrates the *simple case* (left), the *gradient delay case* (middle), and the *gradient error case* (right) for stochastic parallel learning. The simple case is normally from the synchronous parallel learning setting. The gradient delay case is considered for asynchronous parallel learning over sparse feature models (Niu et al., 2011; Mcmahan and Streeter, 2014). Essentially, the gradient delay problem is a simplification of the asynchronous parallel learning problem, and the simplification is from the feature sparseness. For the dense feature models like neural networks, it is unreasonable to use this simplification, and it goes to the gradient error case. As we can see from Figure 1 (right), there are both read-overwrite and write-overwrite problems between the two threads, and this created the gradient error. The gradient error problem is more complex than the gradient delay problem, and it requires new analysis and solutions. We will give more detailed analysis in Section 2.

Although the gradient error problem is more complex, it does not mean that the asynchronous parallel learning is doomed for the dense feature models. We give theoretical analysis to show that the learning process with the gradient error problem can still achieve the optimum given certain conditions, and those conditions are usually valid for the final convergence region of real-world applications.

Based on the analysis, we propose a simple asynchronous parallel learning method for dense feature models including neural networks and other structured models, and it works well in real-world NLP tasks in spite of gradient errors. Base on various dense feature models (LSTM, dense-CRF) and various NLP tasks, experiments show that our method achieves substantial improvement on training speed, and without any loss on accuracy.

2 *AsynGrad*: Asynchronous Parallel Learning with Gradient Error

As we can see from Figure 1, the gradient delay case is mostly considered for sparse feature models (Niu et al., 2011; Mcmahan and Streeter, 2014). Here the read and write of parameters for each sample is fast because the only a very small portion of the features are used for each sample. In this case, the read and write actions are considered being almost atomic actions, and the major concern goes to the “delay” of

Algorithm 1 *AsynGrad*: Asynchronous Parallel Learning with Gradient Error

```
Input: model weights $\boldsymbol{w}$, training set $S$ of $m$ samples
Run $k$ threads in parallel with share memory, and procedure of each thread is as follows:
repeat
    Get a sample $\boldsymbol{z}$ uniformly at random from $S$
    Get the update term $\boldsymbol{s}_{\boldsymbol{z}}(\boldsymbol{w})$, which is computed as $\nabla f_{\boldsymbol{z}}(\boldsymbol{w})$ but usually contains error
    Update $\boldsymbol{w}$ such that $\boldsymbol{w} \leftarrow \boldsymbol{w} - \gamma \boldsymbol{s}_{\boldsymbol{z}}(\boldsymbol{w})$
until Convergence
return $\boldsymbol{w}$
```

the gradients (Niu et al., 2011; Mcmahan and Streeter, 2014). In Figure 1 (middle), thread-1's gradient is delayed by thread-2, because the former is expected to write to the memory before thread-2's write action. The *HogWild* method (Niu et al., 2011) and the delay-tolerant method (Mcmahan and Streeter, 2014) mainly cast/simplify the asynchronous parallel learning problem as the gradient delay problem, and they give solutions accordingly — they show that asynchronous parallel learning can achieve the optimum when dealing with the gradient delay problem.

The gradient delay problem is a simplification of the asynchronous parallel learning problem, and the simplification is from the feature sparseness. For the dense feature models like neural networks, it is no longer reasonable to use the simplification. The major problem for applying asynchronous parallel learning to dense feature models is the gradient errors.

As shown in Figure 1 (right), for dense feature models the read action is together with the gradient-computing action. When the feature is dense, it is not efficient to read all the parameters into a thread before computing the gradient. Also, when the feature is dense, the write action is no longer a fast action. As we can see from Figure 1 (right), there are both read-overwrite and write-overwrite problems, which create gradient errors. The gradient error problem is more complex than the gradient delay problem, and it requires new analysis and solutions.

2.1 *AsynGrad* Algorithm

Let $f(\boldsymbol{w})$ be the objective function of dense feature model and $\boldsymbol{w} \in \mathcal{W}$ is the weight vector. Recall that the SGD update with learning rate γ has a form like this:

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \gamma \nabla f_{\boldsymbol{z}_t}(\boldsymbol{w}_t) \tag{1}$$

where t represents a time stamp, and $\nabla f_{\boldsymbol{z}}(\boldsymbol{w}_t)$ is the stochastic estimation of the gradient based on $\boldsymbol{z}$, which is randomly drawn from the training set S.

Then, we assume a shared memory machine with multiple processors, and a vector of variables $\boldsymbol{w}$ in the shared memory is accessible to all processors. Each processor can read and update $\boldsymbol{w}$.

Based on the multicore computing machine, the proposed method creates k parallel threads, and $\boldsymbol{w}$ is shared among the k threads. For each thread, it gets a sample $\boldsymbol{z}$ uniformly at random from the training set S. Then, it tries to compute the update term $\boldsymbol{s}_{\boldsymbol{z}}(\boldsymbol{w})$ as follows:

$$\boldsymbol{s}_{\boldsymbol{z}}(\boldsymbol{w}) \leftarrow^{\approx} \nabla f_{\boldsymbol{z}}(\boldsymbol{w}) \tag{2}$$

where $\leftarrow^{\approx}$ means the $\boldsymbol{s}_{\boldsymbol{z}}(\boldsymbol{w})$ is computed as the gradient $\nabla f_{\boldsymbol{z}}(\boldsymbol{w})$ in a single thread, but the shared weights $\boldsymbol{w}$ may be partially or completely overwritten by other threads when computing the gradient, which leads to gradient errors of $\nabla f_{\boldsymbol{z}}(\boldsymbol{w})$. Hence, $\boldsymbol{s}_{\boldsymbol{z}}(\boldsymbol{w})$ is an approximation of the gradient $\nabla f_{\boldsymbol{z}}(\boldsymbol{w})$ with potential errors.

Then, the thread updates the shared weights $\boldsymbol{w}$ based on the update term $\boldsymbol{s}_{\boldsymbol{z}}(\boldsymbol{w})$, such that

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \gamma \boldsymbol{s}_{\boldsymbol{z}}(\boldsymbol{w}) \tag{3}$$

For each thread, it repeats this process until it reaches convergence — later we will show that this process can reach convergence by given certain conditions.

To summarize, the proposed parallel learning algorithm called *AsynGrad* is shown in Algorithm 1.

Although the implementation of *AsynGrad* is simple, the method *AsynGrad* itself is not that straightforward, because people may think it is wrong to use a method like *AsynGrad*. There is not much existing theoretical analysis and empirical evidence supporting the design of *AsynGrad*. In this paper, we show that it is actually reasonable and promising to use a method like *AsynGrad*. We give theoretical justification of the convergence of *AsynGrad* based on certain conditions, and we show experimental results that *AsynGrad* indeed works well in practice.

2.2 Theoretical Analysis of *AsynGrad*

Although *AsynGrad* does the weight update in parallel from different threads, from the viewpoint of $\boldsymbol{w}$ it simply receives a sequence of gradient updates (but with potential errors, which is the major difference from non-parallel SGD learning). In other words, the *AsynGrad* learning problem can be casted as a traditional SGD learning problem, and the only difference compared with traditional SGD is that the gradients are with errors. To state our convergence analysis results, we need several assumptions.

We assume f is strongly convex with modulus c, that is, $\forall \boldsymbol{w}, \boldsymbol{w}' \in \mathcal{W}$,

$$f(\boldsymbol{w}') \geq f(\boldsymbol{w}) + (\boldsymbol{w}' - \boldsymbol{w})^T \nabla f(\boldsymbol{w}) + \frac{c}{2}||\boldsymbol{w}' - \boldsymbol{w}||^2 \tag{4}$$

where $||\cdot||$ means 2-norm $||\cdot||_2$ by default in this work. For some dense feature models like dense-CRF, this assumption is suitable for the objective function. For some other dense feature models like neural networks, this assumption is a bit too strong because we know that the objective function of neural networks is non-convex. Nevertheless, even when the objective function is non-convex, the assumption usually holds within the final convergence region because the cost function is locally convex in many practical applications.

We also assume Lipschitz continuous differentiability of ∇f with the constant q, that is, $\forall \boldsymbol{w}, \boldsymbol{w}' \in \mathcal{W}$,

$$||\nabla f(\boldsymbol{w}') - \nabla f(\boldsymbol{w})|| \leq q||\boldsymbol{w}' - \boldsymbol{w}|| \tag{5}$$

Also, let the norm of $\boldsymbol{s_z}(\boldsymbol{w})$ is bounded by $\kappa \in \mathbb{R}^+$:

$$||\boldsymbol{s_z}(\boldsymbol{w})|| \leq \kappa \tag{6}$$

Moreover, it is reasonable to assume

$$\gamma c < 1 \tag{7}$$

because even the ordinary gradient descent methods will diverge if $\gamma c > 1$ (Niu et al., 2011).

Based on the conditions, we show that *AsynGrad* converges closely towards the minimum (optimum) $\boldsymbol{w}^*$ of $f(\boldsymbol{w})$ with a small distance expressed by ϵ, and the convergence rate is given as follows.

Theorem 1 (*AsynGrad* convergence and convergence rate)**.** *With the conditions (4), (5), (6), (7), let $\epsilon > 0$ be a target distance of convergence (i.e., the closeness of the convergence point to the real optimum). Let τ denote the bound to describe the severeness of gradient errors, such that*

$$[\nabla f(\boldsymbol{w}) - \boldsymbol{s}(\boldsymbol{w})]^T(\boldsymbol{w} - \boldsymbol{w}^*) \leq \tau \tag{8}$$

where $\boldsymbol{w}$ is the weight vector during AsynGrad *training, and $\boldsymbol{s}(\boldsymbol{w})$ is expected $\boldsymbol{s_z}(\boldsymbol{w})$ over $\boldsymbol{z}$ such that $\boldsymbol{s}(\boldsymbol{w}) = \mathbb{E}_{\boldsymbol{z}}[\boldsymbol{s_z}(\boldsymbol{w})]$. Let γ be a learning rate as*

$$\gamma = \frac{c\epsilon - 2\tau q}{\beta q \kappa^2} \tag{9}$$

where we can set β as any value as far as $\beta \geq 1$. Let t be the number of updates as follows

$$t \doteq \frac{\beta q \kappa^2 \log{(q a_0/\epsilon)}}{c(c\epsilon - 2\tau q)} \tag{10}$$

Table 1: Some major experimental results on dense-CRF. Time means time cost per iteration.

Method	POS-Tag		Chunking		MSR-WordSeg	
	Acc (%)	Time (s)	F-score (%)	Time (s)	F-score (%)	Time (s)
SGD	97.18	466.08	94.55	1.92	97.13	64.94
AsynGrad(10-thread)	97.18	108.38	94.59	0.25	97.15	8.11
AsynGrad(10-thread)+SR	97.35	25.99	94.60	0.21	97.22	6.59

where $\doteq$ means ceil-rounding of a real value to an integer, and a_0 is the initial distance such that $a_0 = ||\boldsymbol{w}_0 - \boldsymbol{w}^*||^2$. Then, after t updates of $\boldsymbol{w}$, AsynGrad *converges towards the optimum such that $\mathbb{E}[f(\boldsymbol{w}_t) - f(\boldsymbol{w}^*)] \leq \epsilon$, as far as the gradient errors are bounded such that*

$$\tau \leq \frac{c\epsilon}{2q} \tag{11}$$

The proof is in Section 4.

This theorem shows that *AsynGrad* is also convergent towards the optimum of the objective function, as far as the gradient errors are bounded such that Eq.(11) holds. For real-world applications, those assumptions and conditions usually hold at the final convergence region. In the final convergence region, $\boldsymbol{w}$ is not far away from the optimum $\boldsymbol{w}^*$, and both $||\nabla f(\boldsymbol{w})||$ and $||\boldsymbol{s}(\boldsymbol{w})||$ are supposed to be small. Thus, $[\nabla f(\boldsymbol{w}) - \boldsymbol{s}(\boldsymbol{w})]^T(\boldsymbol{w} - \boldsymbol{w}^*)$ is supposed to be small. This makes the bound τ to be small, which makes Eq.(11) valid in the final convergence region.

Moreover, the convergence rate is given in the theorem — *AsynGrad* is guaranteed to converge with t updates, and the value of t is given by Eq.(10).

The theoretical analysis can be concluded as follows. First, it shows that *AsynGrad* is convergent with gradient errors by given certain assumptions and conditions. Second, those assumptions and conditions are usually valid in the final convergence region of practical applications. Third, the convergence speed is given.

3 Experiments

We conduct experiments on natural language processing tasks as follows. Experiments are performed on a computer based on Intel(R) Xeon(R) 3.0GHz CPU of 12 cores, and with 128G memory.

Part-of-Speech Tagging (POS-Tag): We use the standard benchmark dataset in prior work (Collins, 2002), which is derived from PennTreeBank corpus and uses sections 0 to 18 of the Wall Street Journal (WSJ) for training (38,219 samples), and sections 22-24 for testing (5,462 samples). The evaluation metric is per-word accuracy.

Text Chunking (Chunking): The phrase chunking data is extracted from the data of the *CoNLL-2000 shallow-parsing shared task* (Sang and Buchholz, 2000; Sun et al., 2008). The training set consists of 8,936 sentences, and the test set consists of 2,012 sentences. The evaluation metric for this task is balanced F-score.

Word Segmentation (WordSeg). For the LSTM model, we use the social media word segmentation data (Weibo-WordSeg) provided by the NLPCC 2016 Shared Task.The training set contains around 60% of the data set, with 12,081 sentences. The test set contains around 40% of the data set, with 8,055 sentences. However, this data set is relatively new and we do not have good feature templates to implement the dense-CRF model. To deal with this problem, we use the MSR word segmentation data set (MSR-WordSeg) for the experiments on dense-CRF. The MSR-WordSeg data set is provided by *SIGHAN-2004 contest* (Gao et al., 2007). There are 86,918 training samples and 3,985 test samples. For this data set, we have standard feature templates, which are similar to (Sun et al., 2014). The evaluation metric is balanced F-score.

3.1 Experiments on Dense-CRF (Moderately Dense Case)

The dense-CRF is a moderately dense version of CRF by adding rich edge features, which usually has much better accuracy than traditional CRF for NLP tasks (Sun et al., 2012; Sun et al., 2014). For

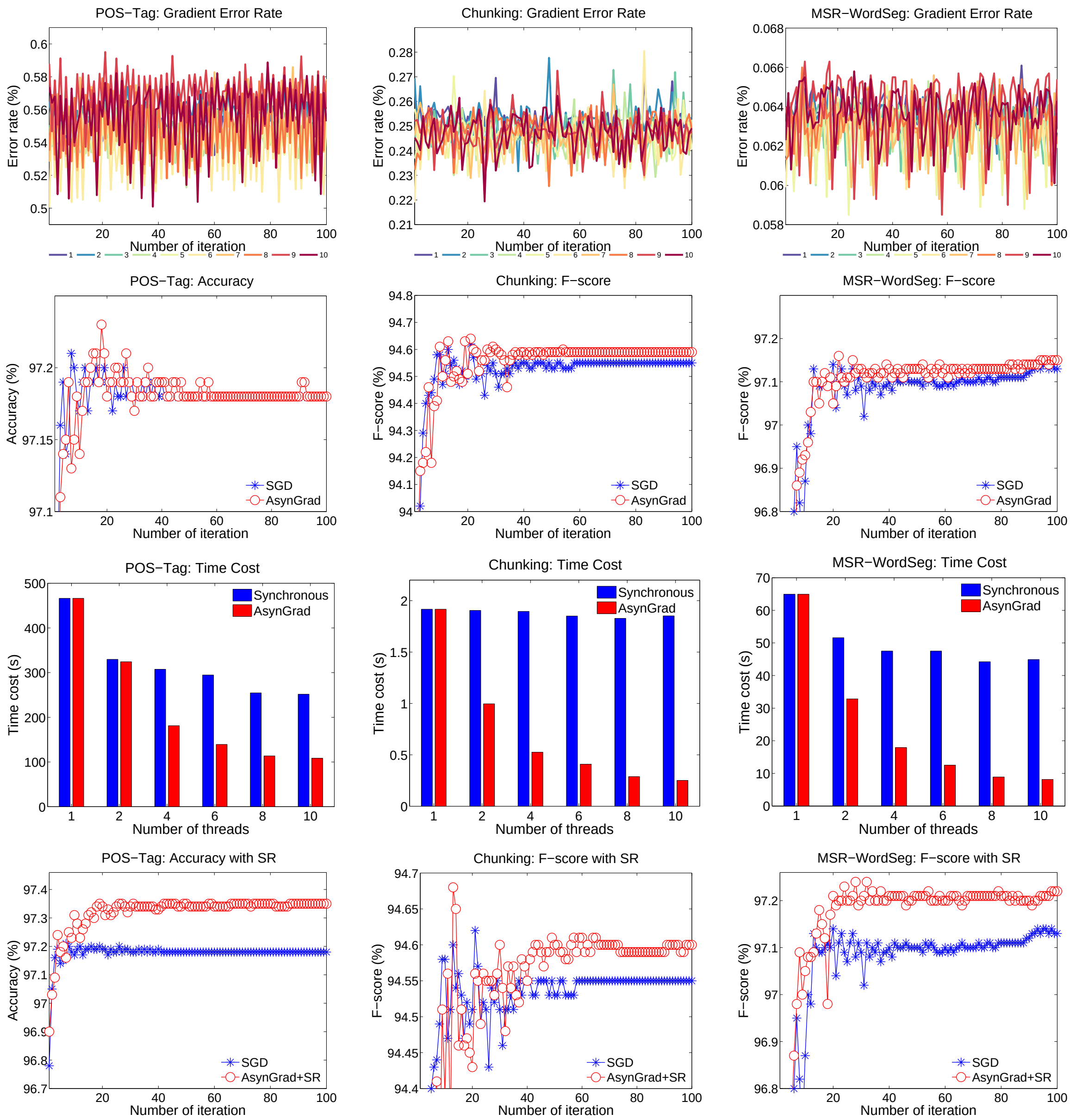

Figure 2: Experimental results on dense-CRF. For the 1st row, 1 to 10 denote the 10 threads of *AsynGrad*. For the 2nd row, *AsynGrad* denotes *AsynGrad* with 10 threads.

traditional implementation of CRF, usually the edges features contain only tag transitions with the form (y_{i-1}, y_i). In dense-CRF, we incorporate local tokens of $\boldsymbol{x}$ into the edge features, which is called rich edge features (Sun et al., 2012; Sun et al., 2014). For example, a rich edge feature can be (x_i, y_{i-1}, y_i) or even $(x_{i-1}, x_i, y_{i-1}, y_i)$. A simple way to automatically create massive rich edge features is to extend each node feature (x, y_i) to the rich edge feature (x, y_{i-1}, y_i), where x means the tokens of a local window. Usually a naive way to do this will cause feature explosion. However, it is easy to solve this problem — only extend high frequency node features (e.g., frequency larger than 10) to rich edge features. In many real-world tasks we find this type of dense-CRF works much better than traditional CRF, and without feature explosion problem even for tasks with many tags (e.g., the POS tagging task).

The standard SGD learning method (Bertsekas, 1999; Bottou and Bousquet, 2008; Shalev-Shwartz and Srebro, 2008) is chosen as the baseline. Based on tuning, the initial learning rate of SGD is set as 0.02, 0.05, 0.1 for the POS-Tag, Chunking, and MSR-WordSeg tasks, respectively. To control overfitting, we use the L_2 regularizer, i.e., a Gaussian prior. The L_2 regularization strength (i.e., the term λ) is set as 1, 0.5, 1 for the three tasks, respectively.

The experimental results are shown in Figure 2. The 1st row shows the percentage of gradient errors

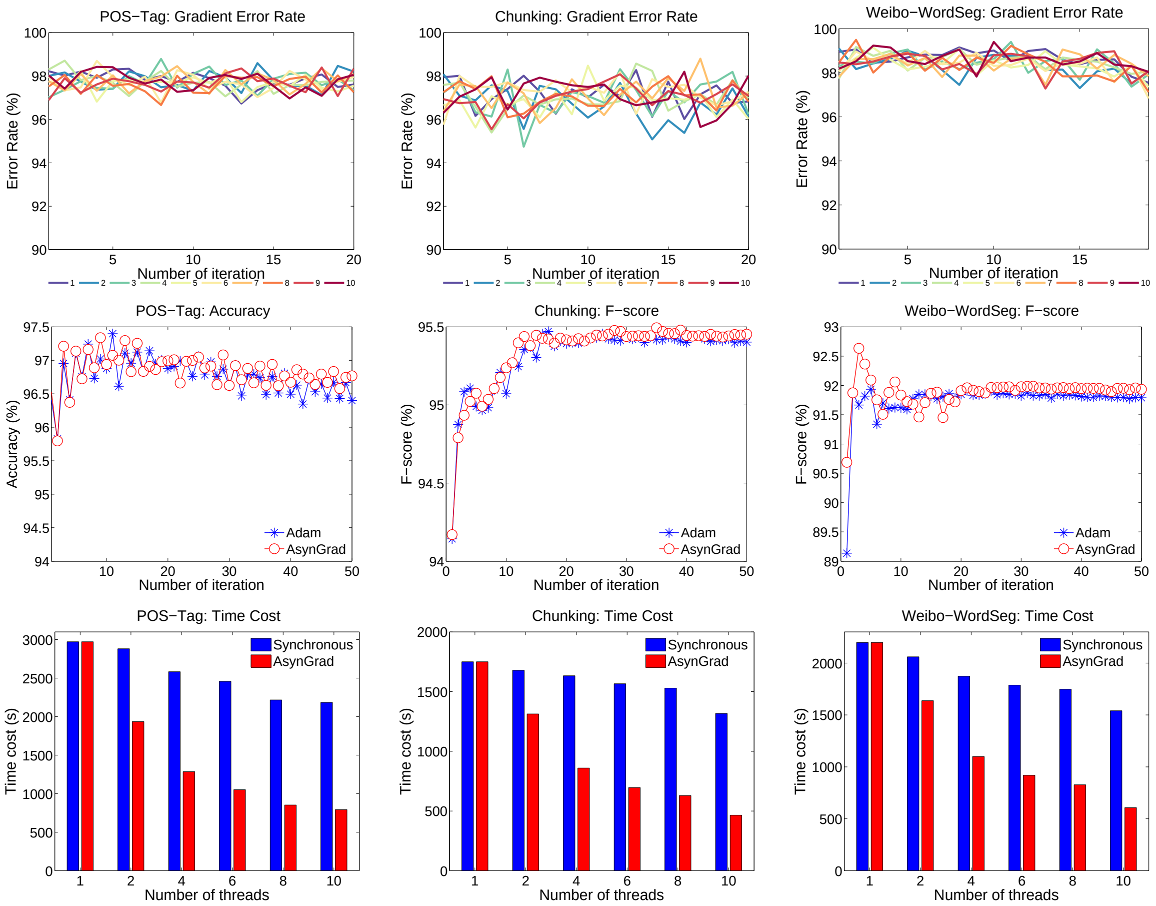

Figure 3: Experimental results on LSTM. For the 1st row, 1 to 10 denote the 10 threads of *AsynGrad*. For the 2nd row, *AsynGrad* denotes *AsynGrad* with 10 threads.

Table 2: Some major experimental results on LSTM. Time means time cost per iteration.

Method	POS-Tag		Chunking		Weibo-WordSeg	
	Acc (%)	Time (s)	F-score (%)	Time (s)	F-score (%)	Time (s)
Adam	96.40	2972.13	95.40	1750.68	91.79	2198.45
AsynGrad(10-thread)	96.77	793.40	95.45	465.95	91.93	607.22

among total gradients. For example, if a thread used 100 gradients and 5 gradients are with errors in this iteration, its gradient error rate is 5%. As we can see, the gradient errors are not rare in asynchronous parallel learning.

The 2nd row shows the accuracy/F-score curves. As we can see, when facing the gradient errors, the proposed asynchronous parallel learning method *AsynGrad* has no loss at all on the accuracy/F-score, compared with traditional SGD (single thread).

The 3rd row shows the speed acceleration of *AsynGrad*, compared with synchronous parallel learning method (Langford et al., 2009). As we can see, when adding more threads for parallel learning, *AsynGrad* achieves substantially better acceleration on the training speed than the synchronous parallel learning method.

The 4th row shows the accuracy/F-score curves by combining *AsynGrad* with structure regularization (SR) (Sun, 2014). SR is a simple method to prevent overfitting by splitting samples into mini-samples, thus it reduces the complexity and sparsity of structures (Sun, 2014). As we can see, *AsynGrad* can easily combine with SR to improve the accuracy/F-score on those tasks.

Some major experimental results are summarized in Table 1.

3.2 Experiments on LSTM (Completely Dense Case)

Recently, long short term memory networks (LSTM) has widely applied to many tasks (Chen et al., 2015; Hammerton, 2003; Cho et al., 2014). In this work, we use the bi-directional long short term memory network (Bi-LSTM) as the implementation of LSTM, which has better accuracy than single-directional LSTM in many practical tasks. The LSTM recurrent cell is controlled by three “gates”, namely input gate, forget gate and output gate.

Adaptive learning versions of SGD are popular to train large neural networks, including the Adam learning method (Kingma and Ba, 2014), the RMSProp learning method (Tieleman and Hinton, 2012), and so on. The experiments on LSTM are based on the Adam learning method, with the hyper parameters as follows: $\beta_1 = \beta_2 = 0.95$, $\epsilon = 1 \times 10^{-4}$ (Kingma and Ba, 2014). The input/hidden dimension is 170, 280, and 220 for the POS-Tag, Chunking, and Weibo-WordSeg tasks, respectively. For the tasks with LSTM, we find there is almost no difference on the results by adding L_2 regularization or not. Hence, we do not add L_2 regularization for LSTM. All weight matrices, except for bias vectors and word embeddings, are diagonal matrices and randomly initialized by normal distribution.

The experimental results are shown in Figure 3. The 1st row shows the percentage of gradient errors among total gradients. As we can see, for neural networks like LSTM, the gradient errors are very common (the rate is over 90%) in asynchronous parallel learning. The gradient error rate is much higher than the dense-CRF case, this is because the LSTM is a much more dense model compared with dense-CRF. In fact, LSTM can be seen as a completely dense model because there is totally no sparse feature existing.

The 2nd row shows the accuracy/F-score curves. As we can see, when facing the intensive gradient errors which are over 90% on very dense models like LSTM, *AsynGrad* has no loss at all on accuracy/F-score.

The 3rd row shows the speed acceleration of *AsynGrad*, compared with synchronous parallel learning method (Langford et al., 2009). As we can see, when adding more threads for parallel learning, *AsynGrad* achieves substantially better acceleration on the training speed than the synchronous version.

The experiments shows that *AsynGrad* can substantially accelerate the training speed of LSTM, and without any loss on accuracy. Some major experimental results are summarized in Table 2.

4 Proof

Here we give the proof of Theorem 1. First, the recursion formula is derived. Then, the bounds are derived.

4.1 Recursion Formula

By subtracting $\boldsymbol{w}^*$ from both sides and taking norms for (1), we have

$$\begin{aligned} ||\boldsymbol{w}_{t+1} - \boldsymbol{w}^*||^2 &= ||\boldsymbol{w}_t - \gamma \boldsymbol{s}_{\boldsymbol{z}_t}(\boldsymbol{w}_t) - \boldsymbol{w}^*||^2 \\ &= ||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 - 2\gamma(\boldsymbol{w}_t - \boldsymbol{w}^*)^T \boldsymbol{s}_{\boldsymbol{z}_t}(\boldsymbol{w}_t) + \gamma^2 ||\boldsymbol{s}_{\boldsymbol{z}_t}(\boldsymbol{w}_t)||^2 \end{aligned} \tag{12}$$

Taking expectations and let $a_t = \mathbb{E}||\boldsymbol{w}_t - \boldsymbol{w}^*||^2$, we have

$$\begin{aligned} a_{t+1} &= a_t - 2\gamma \mathbb{E}[(\boldsymbol{w}_t - \boldsymbol{w}^*)^T \boldsymbol{s}_{\boldsymbol{z}_t}(\boldsymbol{w}_t)] + \gamma^2 \mathbb{E}[||\boldsymbol{s}_{\boldsymbol{z}_t}(\boldsymbol{w}_t)||^2] \\ &\quad \text{(based on (6))} \\ &\le a_t - 2\gamma \mathbb{E}[(\boldsymbol{w}_t - \boldsymbol{w}^*)^T \boldsymbol{s}_{\boldsymbol{z}_t}(\boldsymbol{w}_t)] + \gamma^2 \kappa^2 \\ &\quad \text{(since the random draw of } \boldsymbol{z}_t \text{ is independent of } \boldsymbol{w}_t\text{)} \\ &= a_t - 2\gamma \mathbb{E}[(\boldsymbol{w}_t - \boldsymbol{w}^*)^T \mathbb{E}_{\boldsymbol{z}_t}(\boldsymbol{s}_{\boldsymbol{z}_t}(\boldsymbol{w}_t))] + \gamma^2 \kappa^2 \\ &= a_t - 2\gamma \mathbb{E}[(\boldsymbol{w}_t - \boldsymbol{w}^*)^T \boldsymbol{s}(\boldsymbol{w}_t)] + \gamma^2 \kappa^2 \end{aligned} \tag{13}$$

We define

$$\delta(\boldsymbol{w}) = \nabla f(\boldsymbol{w}) - \boldsymbol{s}(\boldsymbol{w}) \tag{14}$$

and insert it into (4), it goes to

$$\begin{aligned} f(\boldsymbol{w}') &\geq f(\boldsymbol{w}) + (\boldsymbol{w}' - \boldsymbol{w})^T[\boldsymbol{s}(\boldsymbol{w}) + \delta(\boldsymbol{w})] + \frac{c}{2}||\boldsymbol{w}' - \boldsymbol{w}||^2 \\ &= f(\boldsymbol{w}) + (\boldsymbol{w}' - \boldsymbol{w})^T\boldsymbol{s}(\boldsymbol{w}) + \frac{c}{2}||\boldsymbol{w}' - \boldsymbol{w}||^2 + (\boldsymbol{w}' - \boldsymbol{w})^T\delta(\boldsymbol{w}) \end{aligned} \tag{15}$$

By setting $\boldsymbol{w}' = \boldsymbol{w}^*$, we further have

$$\begin{aligned} (\boldsymbol{w} - \boldsymbol{w}^*)^T\boldsymbol{s}(\boldsymbol{w}) &\geq f(\boldsymbol{w}) - f(\boldsymbol{w}^*) + \frac{c}{2}||\boldsymbol{w} - \boldsymbol{w}^*||^2 - (\boldsymbol{w} - \boldsymbol{w}^*)^T\delta(\boldsymbol{w}) \\ &\geq \frac{c}{2}||\boldsymbol{w} - \boldsymbol{w}^*||^2 - (\boldsymbol{w} - \boldsymbol{w}^*)^T\delta(\boldsymbol{w}) \end{aligned} \tag{16}$$

Combining (13) and (16), we have

$$\begin{aligned} a_{t+1} &\leq a_t - 2\gamma\mathbb{E}\Big[\frac{c}{2}||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 - (\boldsymbol{w}_t - \boldsymbol{w}^*)^T\delta(\boldsymbol{w}_t)\Big] + \gamma^2\kappa^2 \\ &= (1 - c\gamma)a_t + 2\gamma\mathbb{E}[(\boldsymbol{w}_t - \boldsymbol{w}^*)^T\delta(\boldsymbol{w}_t)] + \gamma^2\kappa^2 \end{aligned} \tag{17}$$

Considering (8) and (14), it goes to

$$a_{t+1} \leq (1 - c\gamma)a_t + 2\gamma\tau + \gamma^2\kappa^2 \tag{18}$$

We can find a steady state a_∞ by the formula $a_\infty = (1 - c\gamma)a_\infty + 2\gamma\tau + \gamma^2\kappa^2$, which gives

$$a_\infty = \frac{2\tau + \gamma\kappa^2}{c} \tag{19}$$

Defining the function $A(x) = (1 - c\gamma)x + 2\gamma\tau + \gamma^2\kappa^2$, based on (18) we have

$$\begin{aligned} a_{t+1} &\leq A(a_t) \\ &\quad \text{(Taylor expansion of } A(\cdot) \text{ based on } a_\infty \text{, with } \nabla^2 A(\cdot) \text{ being 0)} \\ &= A(a_\infty) + \nabla A(a_\infty)(a_t - a_\infty) \\ &= A(a_\infty) + (1 - c\gamma)(a_t - a_\infty) \\ &= a_\infty + (1 - c\gamma)(a_t - a_\infty) \end{aligned} \tag{20}$$

Thus, we have $a_{t+1} - a_\infty \leq (1 - c\gamma)(a_t - a_\infty)$, and unwrapping it goes to

$$a_t \leq (1 - c\gamma)^t(a_0 - a_\infty) + a_\infty \tag{21}$$

4.2 Bounds

Since $\nabla f(\boldsymbol{w})$ is Lipschitz according to (5), we have

$$f(\boldsymbol{w}) \leq f(\boldsymbol{w}') + \nabla f(\boldsymbol{w}')^T(\boldsymbol{w} - \boldsymbol{w}') + \frac{q}{2}||\boldsymbol{w} - \boldsymbol{w}'||^2$$

Setting $\boldsymbol{w}' = \boldsymbol{w}^*$, it goes to $f(\boldsymbol{w}) - f(\boldsymbol{w}^*) \leq \frac{q}{2}||\boldsymbol{w} - \boldsymbol{w}^*||^2$, such that

$$\mathbb{E}[f(\boldsymbol{w}_t) - f(\boldsymbol{w}^*)] \leq \frac{q}{2}\mathbb{E}||\boldsymbol{w}_t - \boldsymbol{w}^*||^2 = \frac{q}{2}a_t$$

In order to have $E[f(\boldsymbol{w}_t) - f(\boldsymbol{w}^*)] \leq \epsilon$, it is required that $\frac{q}{2}a_t \leq \epsilon$, that is

$$a_t \leq \frac{2\epsilon}{q} \tag{22}$$

Combining (21) and (22), it is required that

$$(1 - c\gamma)^t(a_0 - a_\infty) + a_\infty \leq \frac{2\epsilon}{q} \tag{23}$$

To meet this requirement, it is sufficient to set the learning rate γ such that both terms on the left side are less than $\frac{\epsilon}{q}$. For the requirement of the second term $a_\infty \leq \frac{\epsilon}{q}$, recalling (19), it goes to $\gamma \leq \frac{c\epsilon - 2\tau q}{q\kappa^2}$. Thus, introducing a real value $\beta \geq 1$, we can set γ as

$$\gamma = \frac{c\epsilon - 2\tau q}{\beta q \kappa^2} \tag{24}$$

Note that, to make this formula meaningful, it is required that $c\epsilon - 2\tau q \geq 0$. Thus, it is required that

$$\tau \leq \frac{c\epsilon}{2q}$$

which is solved by the condition of (11).

On the other hand, we analyze the requirement of the first term that $(1 - c\gamma)^t(a_0 - a_\infty) \leq \frac{\epsilon}{q}$. Since $a_0 - a_\infty \leq a_0$, it holds by requiring $(1 - c\gamma)^t a_0 \leq \frac{\epsilon}{q}$, which goes to

$$t \geq \frac{\log \frac{\epsilon}{qa_0}}{\log(1 - c\gamma)} \tag{25}$$

Since $\log(1 - c\gamma) \leq -c\gamma$ given (7), and that $\log \frac{\epsilon}{qa_0}$ is a negative term, we have

$$\frac{\log \frac{\epsilon}{qa_0}}{\log(1 - c\gamma)} \leq \frac{\log \frac{\epsilon}{qa_0}}{-c\gamma}$$

Thus, (25) holds by requiring

$$t \geq \frac{\log \frac{\epsilon}{qa_0}}{-c\gamma} = \frac{\log(qa_0/\epsilon)}{c\gamma} \tag{26}$$

Combining (24) and (26), the problem can be addressed as far as

$$t \geq \frac{\beta q \kappa^2 \log(qa_0/\epsilon)}{c(c\epsilon - 2\tau q)}$$

Thus, setting $t \doteq \frac{\beta q \kappa^2 \log(qa_0/\epsilon)}{c(c\epsilon - 2\tau q)}$ can solve the problem. This completes the proof. □

5 Conclusions

In this paper we show that gradient errors are very common and inevitable in dense feature models. Nevertheless, we show that the learning process with the gradient error problem can still approach the optimum in many practical applications. We propose a simple method *AsynGrad* for asynchronous stochastic parallel learning with gradient error. We show that *AsynGrad* works well for dense feature models like neural networks and dense-CRF, in spite of gradient errors. Base on various NLP tasks, our experiments show that *AsynGrad* can substantially accelerate the training speed of LSTM and dense-CRF, and without any loss on accuracy.

Acknowledgements

Many thanks to Shuming Ma, Jingjing Xu, and Xuancheng Ren for the help on running experiments and drawing figures. This work was supported in part by National Natural Science Foundation of China (No. 61300063), and National High Technology Research and Development Program of China (863 Program, No. 2015AA015404).

References

D.P. Bertsekas. 1999. *Nonlinear Programming*. Athena Scientific.

Léon Bottou and Olivier Bousquet. 2008. The tradeoffs of large scale learning. In *NIPS'08*, volume 20, pages 161–168.

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, Pengfei Liu, and Xuanjing Huang. 2015. Long short-term memory neural networks for chinese word segmentation. In Llu ís M àrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, *EMNLP*, pages 1197–1206. The Association for Computational Linguistics.

KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP'02*, pages 1–8.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large scale distributed deep networks. In *Proceedings of NIPS*, pages 1232–1240.

Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. 2012. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13:165–202.

Jianfeng Gao, Galen Andrew, Mark Johnson, and Kristina Toutanova. 2007. A comparative study of parameter estimation methods for statistical natural language processing. In *Proceedings of ACL'07*, pages 824–831.

Kevin Gimpel, Dipanjan Das, and Noah A. Smith. 2010. Distributed asynchronous online learning for natural language processing. In Mirella Lapata and Anoop Sarkar, editors, *Proceedings of CoNLL*, pages 213–222.

James Hammerton. 2003. Named entity recognition with long short-term memory. In Walter Daelemans and Miles Osborne, editors, *CoNLL*, pages 172–175. ACL.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

John Langford, Alex J. Smola, and Martin Zinkevich. 2009. Slow learners are fast. In *NIPS'09*, pages 2331–2339.

Brendan Mcmahan and Matthew Streeter. 2014. Delay-tolerant algorithms for asynchronous distributed online learning. In *Advances in Neural Information Processing Systems 27*, pages 2915–2923.

Feng Niu, Benjamin Recht, Christopher Re, and Stephen J. Wright. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS'11*, pages 693–701.

Benjamin Recht and Christopher Re. 2013. Parallel stochastic gradient algorithms for large-scale matrix completion. *Math. Program. Comput.*, 5(2):201–226.

Erik Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of CoNLL'00*, pages 127–132.

Shai Shalev-Shwartz and Nathan Srebro. 2008. Svm optimization: inverse dependence on training set size. In *ICML'08*, pages 928–935. ACM.

Xu Sun, Louis-Philippe Morency, Daisuke Okanohara, and Jun'ichi Tsujii. 2008. Modeling latent-dynamic in shallow parsing: A latent conditional model with improved inference. In *Proceedings of COLING'08*, pages 841–848, Manchester, UK.

Xu Sun, Houfeng Wang, and Wenjie Li. 2012. Fast online training with frequency-adaptive learning rates for chinese word segmentation and new word detection. In *Proceedings of ACL'12*, pages 253–262.

Xu Sun, Wenjie Li, Houfeng Wang, and Qin Lu. 2014. Feature-frequency-adaptive on-line training for fast and accurate natural language processing. *Computational Linguistics*, 40(3):563–586.

Xu Sun. 2014. Structure regularization for structured prediction. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2402–2410.

Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5: rmsprop: divide the gradient by a running average of its recent magnitude. In *COURSERA: Neural Networks for Machine Learning*.

Martin Zinkevich, Markus Weimer, Alexander J. Smola, and Lihong Li. 2010. Parallelized stochastic gradient descent. In *Proceedings of NIPS'10*, pages 2595–2603.