Learning Succinct Models: Pipelined Compression with L1-Regularization, Hashing, Elias–Fano Indices, and Quantization

Hajime Senuma[†] [‡] and Akiko Aizawa[‡] [†]

[†]University of Tokyo, Tokyo, Japan
[‡]National Institute of Informatics, Tokyo, Japan
{senuma,aizawa}@nii.ac.jp

Abstract

The recent proliferation of smart devices necessitates methods to learn small-sized models. This paper demonstrates that if there are m features in total but only $n = o(\sqrt{m})$ features are required to distinguish examples, with $\Omega(\log m)$ training examples and reasonable settings, it is possible to obtain a good model in a *succinct* representation using $n \log_2 \frac{m}{n} + o(m)$ bits, by using a pipeline of existing compression methods: L1-regularized logistic regression, feature hashing, Elias–Fano indices, and randomized quantization. An experiment shows that a noun phrase chunking task for which an existing library requires 27 megabytes can be compressed to less than 13 <u>kilo</u>bytes without notable loss of accuracy.

1 Introduction

The age of smart devices has arrived. Cisco reports that, as of 2015, smartphones and tablets account for 15% of IP traffic (against 53% for PCs), and further predicts that, by 2020, this share will have grown to 43%, surpassing the 29% of PCs (Cisco Systems, 2016). This trend increases the need for intelligent processing for these platforms. Hence, the study of statistical methods for natural language processing (NLP) systems on mobile devices has received considerable attention in recent years (Ganchev and Dredze, 2008; Hagiwara and Sekine, 2014; Chen et al., 2015).

Among the various issues in this setting, storage costs pose a particular challenge, because the size of the resulting models often grow quickly. Even a simple noun phrase (NP) chunker can easily take dozens of megabytes if naïvely implemented. Suppose we have m features. A direct implementation then consumes $Z(\mathcal{A}) + O(m\zeta)$, where $Z(\mathcal{A})$ represents the size of an *alphabet* $\mathcal{A}$, or a *feature dictionary* that maps a feature string to an index into its parameter vector, and $O(m\zeta)$ represents the space complexity of a dense real vector as the parameter where using ζ bits to achieve a certain amount of float precision ($64m$ if using double-precision floats). These large-sized models are not only inefficient in terms of network bandwidth, but also significantly increase energy consumption (Han et al., 2016). Therefore, it is vital to devise a method that achieves as small a model as possible.

One of the basic properties of information theory says that, to represent a set of n non-negative integers less than m (or, equivalently, a bit vector of size m containing n 1s), we need at least $B_{m,n} = \lceil \log_2 \binom{m}{n} \rceil \approx n \log_2 \frac{m}{n}$ bits. Recent studies in theoretical computer science show that it is possible to compress a set of non-negative integers while keeping some primitive operations under *succinct* representations, that is, data structures using only $B_{m,n} + o(m)$ bits.

As an analogy to succinct data structures, one may ask a question: if there are m features, but only n of these are useful for distinguishing data, how many bits are required to obtain a good classifier? This paper shows that under several reasonable assumptions such as $n = o(\sqrt{m})$, with $\Omega(\log m)$ examples, it is possible to obtain a model that performs similarly to the original classifier, by using only $B_{m,n} + o(m)$ bits.

To evaluate our method, we implemented conditional random fields (CRFs) by using our pipelined architecture and conducted an experiment on an NP chunking model. The results show that 27 megabytes can be reduced to about 13 kilobytes.

This work is licensed under a Creative Commons Attribution 4.0 International Licence. Licence details: http://creativecommons.org/licenses/by/4.0/

In summary, our work is significant in that

1. we present a pipelined method to acquire a *succinct* model that also has almost the same predictive performance as the model of Ng (2004),
2. although the above method is achieved under some conditions and penalties, the conditions are not ungrounded for NLP tasks and the the penalties are neglible for large m,
3. we also introduce a bitwise trick to perform fast unbiased feature hashing, a part of the pipeline,
4. and our experimentations on sequential labeling tasks achieve smaller models than existing libraries by a factor of one thousand, demonstrating the high practical value of our approach.

This paper is organized as follows. In Section 2, we mention related work. In Section 3, we explain the notation used in the paper. In Section 4, we introduce and analyze our methods. In Section 5, we evaluate our approach by an experiment.

2 Related Work

Ganchev and Dredze (2008) were one of the earliest groups to recognize the importance of acquiring small models for mobile platforms. They also showed that naïve hashing tricks for features (now commonly known as *feature hashing*) can greatly improve the memory efficiency without much loss of accuracy. Shi et al. (2009) showed that feature hashing could be applied to graph models. Weinberger et al. (2009) proposed an unbiased version of feature hashing. Bohnet (2010) applied the method to dependency parsing, and found that it improves not only memory efficiency, but also speed performance. Recent studies (Chen et al., 2015) suggest that hashing tricks also work well in deep neural networks too.

Feature selection through the L1 regularization is also known to be effective, since, roughly speaking, the number of features in the final model results in being logarithmic to the number of total features (Ng, 2004). Online optimization (vis-á-vis batch optimization) with the L1-regularization term gained attention from around 2010 (Duchi and Singer, 2009; Tsuruoka et al., 2009), and with the AdaGrad (Duchi et al., 2011) method becoming particularly popular, because of its theoretical and practical performance.

Compression by quantization is closely related to machine learning. For eaxmple, the Lloyd quantization (Lloyd, 1982) is now used as a popular clustering method known as the K-means clustering. Golovin et al. (2013) showed that simple (simplistic, in fact) quantization techniques actually exhibit good performance in common settings used in NLP; although their methods themselves were basic, they analyzed theoretical effects in detail, and these serve as the key ingredients for our error analysis in Section 4.

The compression of indices is a classical topic in information retrieval (IR). Vigna (2013) introduced the Elias–Fano structure (Fano, 1971; Elias, 1974) into IR, although this was already a popular theme in the succinct data structures community (Grossi and Vitter, 2005; Okanohara and Sadakane, 2007; Golynski et al., 2014). Although other types of succinct structures are common in NLP as well (Watanabe et al., 2009; Sorensen and Allauzen, 2011; Shareghi et al., 2015), the Elias–Fano structure is in that, if regarded as a binary vector, the order of compression ratio depends on the number of 1s (rather than the total length of a vector) and achieves better compression ratio than other succinct models for binary vectors if the ratio of 1s is very low (empirically, below 10%); to put it more concretely, the structure is very attractive when unnecessary indices are culled by some techniques such as L1-regularization. Because of its simplicity and compression ratio, the Elias–Fano structure even gained a certain degree of popularity in industry; for example, as of 2013, it is used as one of the backbones of Facebook's social graph engine (Curtiss et al., 2013). Several variants have been proposed, including sdarry (Okanohara and Sadakane, 2007) and the partitioned Elias–Fano indices (Ottaviano and Venturini, 2014).

Driven by these successes, the study of pipelining compression techniques has flourished in recent years. The web-based morphological parser developed by Hagiwara and Sekine (2014) is similar to our work in its aim and basic scheme, but some parts were not implemented and they gave no theoretical explanation of why pipelined compression would work well. For large-scale neural networks, by using

Han et al. (2015)'s three-step pruning process, Han et al. (2016) gained great empirical success using deep compression, or pipelined compression for a deep neural network with pruning, trained quantization, and Huffman coding.

3 Notation

The base of logarithm is e if not stated explicitly. If the base is x ($x \neq e$), we write $\log_x$. $\mathbb{E}$ denotes an expectation, $\mathbb{R}$ denotes a set of real values, and $\mathbb{N}$ denotes a set of non-negative integers. $\delta_{i,j}$ represents the Kronecker's delta, that is, $\delta_{i,j} = 1$ if $i = j$, and 0 otherwise. $B_{m,n}$ represents the information-theoretic lower bound of encoding a bit vector of size m with n 1s, that is, $n \left\lceil \log_2 \binom{m}{n} \right\rceil$.

4 Learning Succinct Models

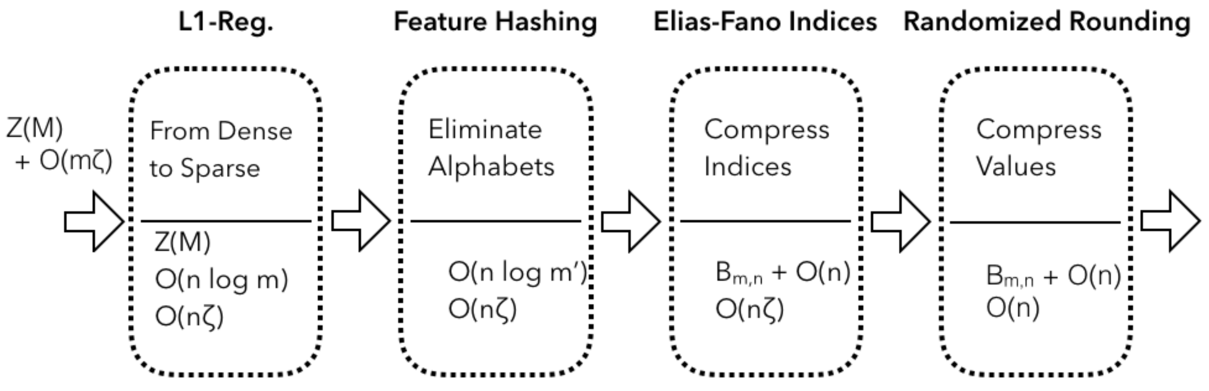

Figure 1: Overview of pipelined compression.

In this section, we present a supervised learning method to obtain a small-sized model for classification tasks. For simplicity, we assume binary classification for each label $y \in \{0, 1\}$ by logistic regression. Suppose there is a set of m features and a dataset $\mathcal{T} = \left\{(\mathbf{x}^{(0)}, y^{(0)}), (\mathbf{x}^{(1)}, y^{(1)}, ..., (\mathbf{x}^{(T-1)}, y^{(T-1)})\right\}$ of T training examples drawn i.i.d. from some distribution $\mathcal{D}$.

Consider the following optimization problem with the L1-regularization term $R(\boldsymbol{\theta}) = ||\boldsymbol{\theta}||_1$

$$\begin{aligned} \max_{\boldsymbol{\theta}} \quad & \sum_{i=0}^{T-1} \log p(y_i \mid \mathbf{x}_i \,; \boldsymbol{\theta}) \\ \text{subject to} \quad & R(\boldsymbol{\theta}) \leq B. \end{aligned} \tag{1}$$

Our goal is to obtain a good parameter $\boldsymbol{\theta}$ that minimizes the expected logistic log loss, that is,

$$\varepsilon(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[-\log p(y \mid \mathbf{x} \,; \boldsymbol{\theta})] \tag{2}$$

with as small a model as possible. We also define the empirical loss

$$\hat{\varepsilon}(\boldsymbol{\theta}) = \hat{\varepsilon}_{\mathcal{T}}(\boldsymbol{\theta}) = \frac{1}{T} \sum_{i=0}^{T-1} -\log p(y_i \mid \mathbf{x}_i \,; \boldsymbol{\theta}). \tag{3}$$

The m-dimensional vector $\boldsymbol{\theta}$ is often implemented as a float array of size m, consuming $O(m\zeta)$ bits. Concretely, it occupies $64m$ bits if we use double-precision ($\zeta = 64$) and $32m$ bits if single-precision ($\zeta = 32$). As m can be millions, billions, or more in NLP tasks, this size complexity is often non-negligible for mobile and embedded devices (e.g., 762.9 megabytes for 100,000,000 features when using double-precision).

In addition to the parameter vector, we need to store an *alphabet* or a *feature dictionary* $\mathcal{A}\colon \mathbb{N} \to \{0, ..., m-1\}$ such that $\mathcal{A}(\mathfrak{f}_i) = i$ for $\mathfrak{f}_0, ..., \mathfrak{f}_{m-1} \in \mathbb{N}$. For example, if the 12-th element in a feature set represents "the current token is `appropriate`, the previous token is `I`, and the current label is `VERB`", one possible formulation is $\mathfrak{f}_{12}$ = `utf8("bigram[-1:0]=I:appropriate&label=VERB")`, where `utf8` denotes a non-negative integer given by the UTF-8 bit representation of a string. A weight value for this feature can then be accessed as $\boldsymbol{\theta}_{\mathcal{A}(\mathfrak{f}_{12})}$. There are two problems with the use of alphabets. First, they consume too much storage space (Ganchev and Dredze, 2008). Second, if implemented with a hash table (as is often the case), they are so slow that they become the bottleneck of the entire learning system (Bohnet, 2010). In the following, $Z(\mathcal{A})$ denotes the size of an alphabet $\mathcal{A}$.

4.1 Pipelined method

Let us now introduce a pipelined method to reduce both $O(m\zeta)$ and $Z(\mathcal{A})$.

Definition 4.1. Given inputs $\mathcal{T}$,

1. Train the model by using L1-regularization according to the method of Ng (2004):
   (a) Divide the data $\mathcal{T}$ into a training set $\mathcal{T}_1$ of the size $(1-\gamma)T$ and a development set $\mathcal{T}_2$ of the size γT, for some $\gamma \in \mathbb{R}$.
   (b) For $B = 0, 1, 2, 4, ..., C$, solve the optimization under each of these hyperparameters with $\mathcal{T}_1$, giving $\boldsymbol{\theta}_0, ..., \boldsymbol{\theta}_C$.
   (c) Choose $\boldsymbol{\theta} = \arg\min_{\{i \in 0,1,2,...,C\}} \hat{\varepsilon_{\mathcal{T}_2}}(\boldsymbol{\theta}_i)$, or the best model for the development data set.

   As we performed the L1-regularization, it is safe to assume that the non-zero components are quite small. $Z(\mathcal{A}) + O(n \log m) + O(n\zeta)$; $Z(\mathcal{A})$ for the alphabet, $\lceil \log m \rceil$ bits for each of the n indices, and $zeta$) bits for each of the n values.

2. Eliminate the alphabet and reduce the dimensionality of vectors by using the *unbiased feature hashing* (Weinberger et al., 2009). Given two hash functions $h\colon \mathbb{N} \to \{0, ..., m'-1\}$ and $\xi\colon \mathbb{N} \to \{-1, 1\}$, we define the (unbiased) *hashed feature map* ϕ such that $\phi^{(h,\xi)(\mathbf{x})} = \sum_{j\colon h(j)=i} \xi(i)\mathbf{x}_i$. We apply this ϕ to the parameter vector and all future input vectors. As this map acts as a proxy for the alphabet, we no longer need to store the alphabet. As a side-effect, if we choose $m' < m$, the dimensionality is reduced from m to m'. Note that this is said to be unbiased because given $\langle \mathbf{x}, \mathbf{x}' \rangle_\phi := \langle \phi^{(h,\xi)}(\mathbf{x}), \phi^{(h,\xi)}(\mathbf{x}') \rangle$, $\mathbb{E}\langle \mathbf{x}, \mathbf{x}' \rangle_\phi = \langle \mathbf{x}, \mathbf{x}' \rangle$. In other words, on average, prediction using a hashed map space is the same as the original version.

   At this point, the size has been reduced to $O(n \log m') + O(n\zeta)$; $\lceil \log m' \rceil$ for each of the n indices and ζ) bits for each of the n values.

3. Compress the set of n indices by using the Elias–Fano scheme (Fano, 1971; Elias, 1974). In brief, n integers are compressed to $n\lceil \log_2 m/n \rceil + f(n)$, where $f(n) = O(n)$ and $f(n) \le 2Sn$ for some speed-memory trade-off hyperparameter $1 \le S < 2$ (usually less than 1.01). We omit the details here, but interested readers may consult Vigna (2013)'s good introduction.

   At this point, the size has been reduced to $n\lceil \log_2 m'/n \rceil + O(n) + O(n\zeta)$.

4. Compress the set of values by using the *unbiased randomized rounding* (Golovin et al., 2013). Let μ and ν be some non-negative integers, then it encodes each value as a fixed-point number with Q$\mu.\nu$ encoding, that is, as μ integer bits and ν fractional bits (e.g., if $\mu = \nu = 2$, we can represent a maximum of $(11.11)_2 = 3.75$). Including a sign bit, $\mu + \nu + 1$ for each value. μ and ν are chosen so that they satisfy the conditions of the error anaysis in the following subsection.

   At this point, the size has been reduced to $n\lceil \log_2 m'/n \rceil + O(n)$.

Overall, because $n\lceil \log_2 m/n \rceil = B_{m,n} + O(n)$ (Golynski et al., 2014), choosing $m' < m$ and assuming $n = o(m)$ achieves a model with $B_{m,n} + o(m)$. In the next subsection, however, we assume tighter conditions such as $n = o(\sqrt{m})$ to achieve good error bounds.

4.2 Error analysis

Each compression technique (except Elias–Fano) adds several penalties to the predictive performance. Therefore, intuitively, pipelined compression should lower the performance significantly. Contrary to such intuition, however, the increase in the error is negligible under certain circumstances.

Theorem 4.2. *Suppose that there exist m features in total, but only n features are important, so that the optimal parameter vector $\theta^* \in \mathbb{R}^m$ has only n non-zero components with indices $0 \le i_0, i_1, ..., i_{n-1} < m$. Further, assume that each non-zero component is bounded by some constant no less than 1, that is, $0 < |\theta_{i_j}| \le K$ for $j = 0, ..., n-1$ with $K \ge 1$.*

Let us be given a training data set $\mathcal{T} = \left\{(\mathbf{x}^{(0)}, y^{(0)}), (\mathbf{x}^{(1)}, y^{(1)}, ..., (\mathbf{x}^{(T-1)}, y^{(T-1)})\right\}$ of T training examples drawn i.i.d. from some distribution $\mathcal{D}$. Assume that the number of non-zero components of each input is always bounded by some non-negative constant much smaller than m, and that n is also much smaller than m as follows:

$$||\mathbf{x}||_0 \le k \text{ for any } (\mathbf{x}, y) \sim \mathcal{D}, k \ge 0, k = o(\sqrt{m}), n = o(\sqrt{m}), \text{ and } K = o(m). \tag{4}$$

Then, with probability at least $(1-\delta)(1+f_1(m))$ where $f_1(m) = o(1)$, it is possible to obtain a model with size (in bits) at most

$$B_{m,n} + o(m) \tag{5}$$

with errors to the expected logistic loss

$$\varepsilon(\hat{\theta}) \le \epsilon_{mul}\left(\varepsilon(\theta^*) + \epsilon_{add}\right) \tag{6}$$

where $\epsilon_{mul} = 1 + f_2(m)$ with f_2 such that $f_2(m) = o(1)$, by performing $C \ge nk$ iterations to find a good hyperparameter and by using the T examples such that

$$T = \Omega((\log m) \cdot poly(n, K, \log(1/\delta), 1/\epsilon_{add}, C)). \tag{7}$$

Prior to its proof, let us explain what the above theorem says. If we omit the conditions of Equation (4), we have the same setting used in Ng (2004, Theorem 3.1). Thus, if we perform L1-regularized learning as defined in 1 of Definition 4.1, Ng's theorem implies that Equations (6) and (7) hold, satisfying $f_1(m) = 0$ and $f_2(m) = 0$.

In other words, the above theorem implies that if we add the conditions defined in Equation (4)) to Ng's theorem, it is possible to obtain a succinct representation (Equation (5)) whose predictive performance is nearly as good as that of Ng's theorem, albeit with a probabilistic penalty $f_1(m)$ and an error penalty $f_2(m)$. However, the penalties are negligible because $f_1(m) = o(1)$ and $f_2(m) = o(1)$.

Note that for NLP tasks, some of the conditions can be justified to a certain extent by the power law. Let us follow the discussion presented by Duchi et al. (2011, Section 1.3). In NLP tasks, the appearance of features often follows the power law. Thus, we can assume that the i-th most frequent feature appears with probability $p_i = \min(1, ci^{-\alpha})$ for some $a \in (1, +\infty)$ and a positive constant c. Given $\mathbf{x}$, $\mathbb{E}||\mathbf{x}||_0 \le c\sum_{i=1}^{m} i^{-\alpha/2} = O(\log d)$ for $\alpha \ge 2$ and $||\mathbf{x}||_0 \le O(m^{1-\alpha/2})$ for $\alpha \in (1, 2)$. Thus $\mathbb{E}||\mathbf{x}||_0 = o(\sqrt{m})$ for any $\mathbf{x}$ and $\alpha \in (1, +\infty)$. This assumption and derivation by Duchi et al. imply that the condition $k = o(\sqrt{m})$ is not ungrounded. The same discussion applies to $n = o(\sqrt{m})$ too. The condition $K = o(m)$ is less obvious than the other two, but we often assume a good parameter does not have any excessively large values.

We now prove the above theorem.

Proof. If we omit the conditions of Equation (4)), then the method in 4.1.1 can be used to show that Equations (6) and (7) hold, satisfying $\epsilon_{mul} = 1$ (Ng, 2004, Theorem 3.1).

Next, we apply the unbiased feature hashing defined in 4.1.2. As the expected dot-product between vectors in a reduced space is the same as the original, this does not change the expected error (Weinberger et al., 2009). There is, however, one problem: hash collisions. When a collision occurs for an input $\mathbf{x}$, for the vector converted from $\mathbf{x}$, the absolute value for some index may exceed 1. However, the error analysis of randomized rounding requires the absolute value of any index to be bounded by 1. Hence, applying feature hashing before randomized rounding can break the subsequent analysis. Thankfully, if we use $h : \mathbb{N} \to 0, ..., m'$ such that $||\mathbf{x}||_0 = o(m')$ for any $\mathbf{x} \in \mathcal{D}$, with *high probability* (that is, $1 - o(1)$), collisions never occur. This is a well-known fact resulting from Sterling's approximation (in the following , f, g, h, i are $o(1)$ and irrelevant to the rest of this proof).

$$\begin{aligned}
\binom{n}{k} &= \frac{n!}{k!(n-k)!} = \frac{(1+f(n))\sqrt{2\pi n}e^{-n}n^n}{k!(1+g(n-k))\sqrt{2\pi(n-k)}e^{-(n-k)}(n-k)^{n-k}} \\
&= \frac{(1+f(n))}{(1+g(n-k))} \cdot \frac{1}{k!} \cdot \left(\frac{n}{n-k}\right)^{1/2} e^{-k} \left(\frac{n}{n-k}\right)^{-k} \left(\frac{n}{n-k}\right)^{n} n^k \\
&= \frac{(1+f(n))}{(1+g(n-k))} \cdot \frac{n^k}{k!} \cdot \left(1-\frac{k}{n}\right)^{k-1/2} e^{-k} \left(1-\frac{k}{n}\right)^{-n} \\
&= \frac{(1+f(n))}{(1+g(n-k))} \cdot \frac{n^k}{k!} \cdot (1+o(1))e^{-k} \frac{1}{\left(1-\frac{k}{n}\right)^n} \\
&= \frac{(1+f(n))(1+h(k))}{(1+g(n-k))(1+i(n))} \cdot \frac{n^k}{k!} \cdot e^{-k} \cdot \frac{1}{e^{-k}} = \frac{(1+f(n))(1+h(k))}{(1+g(n-k))(1+i(n))} \cdot \frac{n^k}{k!}
\end{aligned}$$

Using this equation, it is possible to estimate the hash collision ratio. If we assume Equation (4), $m' = c\sqrt{m}$ for some positive c satisfies the condition we mentioned above.

We then apply the Elias–Fano structure defined in Definition 4.1.3. Because this is a lossless compression method for indices, the error analysis remains the same. The remaining problem is how to store the values in $o(m)$ bits.

Finally, we apply the unbiased randomized rounding as in Definition 4.1.4. Golovin et al. (2013, Theorem 4.3) implies that if the value of each non-zero component of θ^* is bounded by some K and the number of non-zero components of any input is bounded by some k such that $K \geq 1$ and $k \geq 0$, then it is possible to derive a new parameter vector $\hat{\theta}$, with each value using $(1 + \mu + \nu)$ bits, such that $\varepsilon(\hat{\theta}) \leq \epsilon_{mul} \cdot \varepsilon(\theta^*)$, where

$$0 \leq \epsilon_{mul} \leq 2\chi\sqrt{2\pi k}\exp\left(\frac{\chi^2 n}{2}\right), \tag{8}$$

$\mu = \lceil \log_2 K \rceil (= o(\log m)$ if $K = o(m))$, and $\nu = -\log_2 \chi$ (so $\chi = \frac{1}{2^\nu}$). If we use $\nu = \lceil cm^{1/2} \rceil (= \Theta(\sqrt{m}))$ for some positive constant c, assuming the conditions (4) hold, $\epsilon_{mul} = f_2(m)$ with f_2 such that $f_2(m) = o(1)$. Here $f_2(m) = o(1)$ holds because $\lim_{m\to+\infty} \chi\sqrt{k} = \lim_{m\to+\infty} \sqrt{k \cdot \chi^2} = \lim_{m\to+\infty} \sqrt{o(\sqrt{m})/O(2^{m^{1/2}})} = 0$ and the same limit also applies to the $\chi^2 n$ term.

Thus, to store the values, we need $n(1 + \mu + \nu) = o(\sqrt{m}) \cdot (1 + o(\log m) + \Theta(\sqrt{m})) = o(m)$ bits in total. This completes the proof. □

4.3 Practical settings

In practical situations, several violations of the method can make it more efficient.

For example, as presented by Weinberger et al. (2009), it is possible to reduce both storage and RAM usage during training by applying feature hashing prior to training with L1-regularization. In this case, it is not clear whether the above theorem holds, as we solve the optimization problem in the reduced-dimension space rather than the original one. Nevertheless, the predictive performance of resulting models will not change significantly. Weinberger et al. (2009, Corollary 5) pointed out that the number of instances enters logarithmically into the analysis of the maximal canonical distortion. Additionally, Ng (2004) showed that the number of features m affects the number of required instances only logarithmically. Combining these results implies that, if we carefully choose informative instances for training, the distortion in the reduced space is only affected by $\log\log m$, which should be negligible. Thus, we expect the theorem to virtually hold even in this setting.

Another issue involves solving Equation (1). Several concrete optimization algorithms such as AdaGrad (Duchi et al., 2011) solve the dual problem of Equation (1), that is, $\arg\max_{\boldsymbol{\theta}} \sum_{i=0}^{T-1} \log p(y_i|\mathbf{x}_i;\boldsymbol{\theta}) - \lambda R(\boldsymbol{\theta})$ where $\lambda \propto \frac{1}{B}$ for $B > 0$. This is said to be dual because, for any $\boldsymbol{\theta}$ obtained by Equation (1) with some hyperparameter B, there is exactly one hyperparameter λ that gives the same $\boldsymbol{\theta}$. Unfortunately, it is often impossible to get the exact relationship between the hyperparameters B and λ. However, because $\lambda \propto \frac{1}{B}$, a good value may be found by trying $\lambda = 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, ...$

4.4 Bitwise feature hashing

On first examination, unbiased feature hashing (Weinberger et al., 2009) may appear to be twice as costly as the naïve version (that is, always $\xi(i) = 1$ in Definition 4.1.2), as it requires two hash functions to be computed: h and ξ. Actually, by leveraging some bitwise operations, we need only one hash computation and just another three CPU cycles per feature to perform the unbiased version. Let us consider the following algorithm to convert k items of an original index–value pair to the same number of items of pairs under feature hashing. In the algorithm, `>>` denotes an arithmetic right shift (also called a signed right shift).

```
Data: A 2-tuple of indices and values $(I; V)$ such that $I \in \mathbb{N}^k$ and $V \in \mathbb{R}^k$ for some $k$.
      Dimensionality $m'$ satisfying $m' = 2^a$ for some $a \in \mathbb{N}$.
Result: A 2-tuple of indices and values $(I^*; V^*)$ such that $I^* \in \mathbb{N}^k$ and $V^* \in \mathbb{R}^k$ for some $k$ and
        $0 \leq i' < m' < 2^{32}$ for any $i \in I^*$.
1 $MASK \leftarrow m' - 1$ ;
2 for int $i = 0$; $i < k$; $i \leftarrow i + 1$ do
3     $t \leftarrow h(I_i)$ ;
4     sign $\leftarrow (t$ >> $31) \mid 1$ ;
5     $I_i^* \leftarrow t$ & $MASK$ ;
6     $V_i^* \leftarrow sign \cdot V_i$ ;
7 end
8 return $(I^*; V^*)$ ;
```

Algorithm 1: Bitwise unbiased feature hashing

The improvements over the original version are:

1. the original version requires two hash functions, whereas our version requires only one,
2. the resulting index–value pairs are not merged—because the dot-product between two vectors is distributive ($a \cdot (b + c) = a \cdot b + a \cdot c$), there is no need to merge indices—so there is no need to use slow data structures such as hash maps for merging, and
3. compared with Bohnet (2010)'s approach, which uses a `%` (modulo) operator to gain the desired dimension, our approach uses `&` (bitwise mask), so it runs faster in common architectures. The downside is that the resulting dimension will be limited to a power of two.

5 Experimental Results

To evaluate our approach, we implemented a linear-chained conditional random field (CRF) (Lafferty et al., 2001; Sha and Pereira, 2003) and tested it on an NP chunking task. For the optimization problem, we used the AdaGrad, an online optimization algorithm, with the L1-regularization term by diagonal primal-dual subgradient update (Duchi et al., 2011).

5.1 Implementation

We implemented our CRF library with ECMAScript 6, but its core component for optimization computation was handwritten by asm.js[1], a subset of ECMAScript 6 (that is, so-called JavaScript 6). The language

[1]".js" is not a file extension, but a part of the name of the language.

was designed mainly by Mozilla as a performance improvement technique for web platforms. Although it is conformant to ECMAScript 6, the language is actually lower-level than C, and Mozilla claims it is only at most twice as slow as native code. We used the language because, by using a JavaScript-compatible language, it is possible to embed the resulting classifier in web pages for use as a fundamental library in web systems, not only on desktop machines but also on mobile platforms.

5.2 NP chunking

μ \ ν	3	2	1
3	0.9720	0.9703	0.9621
2	0.9673	0.9681	0.9610
1	0.9470	0.9460	0.9392

Table 1: Macro-averaged F1 values under various μ and ν values for $Q\mu.\nu$ encoding.

To test our CRF implementation, following Sha and Pereira (2003), we performed an NP chunking task using the CoNLL-2000 text chunking task data (Tjong Kim Sang and Buchholz, 2000). In the shared task data, the labels B-NP and I-NP were retained, but all other labels were converted to O. Therefore, this is basically a sequential labeling problem with three labels. The features are the same as those used by Sha and Pereira (2003), except that they reformulated two consecutive labels as a new one, whereas we used the original labels. The total number of features is 1,015,621.

The training dataset consists of 8,936 instances, and the test dataset contains 2,012. Of the training data set, we used 7,148 instances (4/5) for training and the remaining 1,788 instances as development data.

For feature hashing, we used 2^{20} as the resulting dimension. There are three hyperparameters for AdaGrad: δ, η, and λ. For the first two, we simply used $\delta = \eta = 1.0$. To find a good value for λ, or the coefficient of the regularization term, we tried $\{1.0, 0.5, 0.25, 0.125, ..., \}$, and found that $\lambda = 0.5^{14} \sim 0.00006104$ gave the best results, with a macro-averaged F1 score of 0.9722. The number of active features (L0-norm) was 8,824. We also tested various μ and ν for the unbiased randomized rounding, and found that $\mu = \nu = 3$ gave a macro-averaged F1 score of 0.9720, which is slightly less than the original. In this case, the number of active features further decreased to 6,785. Overall, the size of our model was compressed to 12,745 bytes.

We compared our implementation with CRFSuite 1.2, a popular linear-chained CRF library written in C++ (Okazaki, 2007). The library has a convenient functionality that allows strings to be used as feature IDs. The downside is that the resulting model tends to be large, because it requires an alphabet to be stored. Using this library, the obtained model had a size of 27 megabytes, and achieved a macro-averaged F1 score was 0.973009.

In summary, although our method is inferior to the existing implementation by 0.001 in terms of F1 score, the size efficiency is more than 2,000 times better.

5.3 Performance of bitwise feature hashing

We also tested the performance of the bitwise feature hashing described in Section 4.4. For comparison, we prepared three functions as follows.

1. *Unbiased* feature hashing. The algorithm is similar to the one given by Bohnet (2010) but we employed two hash functions to implement the unbiased version and added values when collisions occurred, whereas Bohnet's method just overwrites the previous value. We used JavaScript's built-in object type as a dictionary (or hash map).

2. *Non-merge* feature hashing. This is almost identical to the standard version, but instead of using a dictionary (or hash map) structure, it emits the resulting index–value pairs into array structures

without merging the indices. The code was implemented without asm.js tuning, as this does not leverage bitwise operations.

3. *Bitwise* feature hashing with asm.js tuning (Section 4.4).

The environment used in this experiment was OS X 10.11.6 with 2.2 GHz Intel Core i7. The code was micro-benchmarked by Benchmark.js 2.1.0, developed by Mathias Bynens [2], under the following web browsers: Apple Safari 10.0 (with the JavaScript engine JavaScriptCore, also known as Nitro), Google Chrome 53.0.2785.143 (V8), and Mozilla Firefox 49.0.1 (SpiderMonkey). The experimental results are shown below.

	Standard	Non-merge	Bitwise
Apple Safari 10	1,108	3,426	**6,670**
Google Chrome 53	506	2,524	**4,710**
Mozilla Firefox 49	239	737	**4,446**

Table 2: Performance of each feature hashing implementation. Higher is better. Unit: 1k ops/sec.

The results in this table show that the bitwise version is more than five times faster than the standard one in all environments.

As we see, Safari was faster than the other browsers, possibly because the code was run under Apple's OS. Firefox was slightly slower than Chrome in general settings, but with asm.js enabled it achieved almost the same performance.

In conclusion, the bitwise feature hashing can offer powerful performance improvement when the task involves a number of features and processing those features is the bottleneck of the learning system.

6 Conclusions

This paper showed that pipelining compression methods for machine learning can achieve a succinct model, both theoretically and practically. Although each technique is classical and well-studied, our result is significant in that these techniques actually complement each other, leading to drastic compression without harming the predictive power. We also examined bitwise feature hashing, a subtle but powerful modification for improving processing performance. In future work, we will try to incorporate other compression techniques to build various pipeline configurations and compare the performance of each configuration.

Acknowledgements

This work was supported by CREST, Japan Science and Technology Agency (JST). We are grateful to three anonymous reviewers for their helpful comments.

[2]`https://github.com/bestiejs/benchmark.js`

References

Bernd Bohnet. 2010. Very High Accuracy and Fast Dependency Parsing is not a Contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING '10)*, pages 89–97.

Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. 2015. Compressing Neural Networks with the Hashing Trick. *Proceedings of the 32nd International Conference on Machine Learning*, pages 2285–2294, apr.

Cisco Systems. 2016. The Zettabyte Era: Trends and Analysis. Technical report.

Michael Curtiss, Iain Becker, Tudor Bosman, Sergey Doroshenko, Lucian Grijincu, Tom Jackson, Sandhya Kunnatur, Soren Lassen, Philip Pronin, Sriram Sankar, Guanghao Shen, Gintaras Woss, Chao Yang, and Ning Zhang. 2013. Unicorn: A System for Searching the Social Graph. *Proceedings of the VLDB Endowment*, 6(11):1150–1161, aug.

John Duchi and Yoram Singer. 2009. Efficient Online and Batch Learning using Forward Backward Splitting. *Journal of Machine Learning Research*, 10:2873–2898.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Peter Elias. 1974. Efficient Storage and Retrieval by Content and Address of Static Files. *Journal of the ACM*, 21(2):246–260.

Robert M. Fano. 1971. On the Number of Bits Required to Implement An Associative Memory. Technical report, Memorandum 61, Computer Structures Group, MIT, Cambridge, MA.

Kuzman Ganchev and Mark Dredze. 2008. Small Statistical Models by Random Feature Mixing. In *Proceedings of the ACL08 HLT Workshop on Mobile Language Processing*, pages 19–20.

Daniel Golovin, D. Sculley, H. Brendan McMahan, and Michael Young. 2013. Large-Scale Learning with Less RAM via Randomization. *Proceedings of the 30th International Conference on Machine Learning*, 28:325–333, mar.

Alexander Golynski, Alessio Orlandi, Rajeev Raman, and S. Srinivasa Rao. 2014. Optimal Indexes for Sparse Bit Vectors. *Algorithmica*, 69(4):906–924, aug.

Roberto Grossi and Jeffrey Scott Vitter. 2005. Compressed Suffix Arrays and Suffix Trees with Applications to Text Indexing and String Matching. *SIAM Journal on Computing*, 35(2):378–407, jan.

Masato Hagiwara and Satoshi Sekine. 2014. Lightweight Client-Side Chinese/Japanese Morphological Analyzer Based on Online Learning. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: System Demonstrations*, pages 39–43.

Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning bothWeights and Connections for Efficient Neural Networks. In *Advances in Neural Information Processing Systems 28*, pages 1135–1143.

Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations*.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional Random Fields : Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289.

Stuart P. Lloyd. 1982. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137.

Andrew Y. Ng. 2004. Feature selection, L1 vs. L2 regularization, and rotational invariance. In *Proceedings of the 21 st International Conference on Machine Learning*.

Daisuke Okanohara and Kunihiko Sadakane. 2007. Practical Entropy-Compressed Rank / Select Dictionary. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 60–70.

Naoaki Okazaki. 2007. CRFsuite: a fast implementation of conditional random fields (CRFs).

Giuseppe Ottaviano and Rossano Venturini. 2014. Partitioned Elias-Fano Indexes. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 273–282.

Fei Sha and Fernando Pereira. 2003. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, number June, pages 134–141, Morristown, NJ, USA. Association for Computational Linguistics.

Ehsan Shareghi, Matthias Petri, Gholamreza Haffari, and Trevor Cohn. 2015. Compact, Efficient and Unlimited Capacity: Language Modeling with Compressed Suffix Trees. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2409–2418.

Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and S.V.N. Vishwanathan. 2009. Hash Kernels for Structured Data. *Journal of Machine Learning Research*, 10:2615–2637.

Jeffrey Sorensen and Cyril Allauzen. 2011. Unary Data Structures for Language Models. In *INTERSPEECH 2011*, pages 1425–1428.

Erik F. Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the CoNLL-2000 Shared Task: Chunking. In *Proceedings of CoNLL-2000 and LLL-2000*.

Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. 2009. Stochastic Gradient Descent Training for L1-regularized Log-linear Models with Cumulative Penalty. In *Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP*, pages 477–485.

Sebastiano Vigna. 2013. Quasi-Succinct Indices. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining - WSDM '13*, pages 83–92, New York, New York, USA. ACM Press.

Taro Watanabe, Hajime Tsukada, and Hideki Isozaki. 2009. A Succinct N-gram Language Model. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 341–344.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature Hashing for Large Scale Multitask Learning. In *Proceedings of the 26th International Conference on Machine Learning*.