GCWSNet: Generalized Consistent Weighted Sampling for Scalable and Accurate Training of Neural Networks

Ping Li
LinkedIn Corporation
700 Bellevue Way NE, Bellevue WA, 98004, USA
plini@linkedin.com

Weijie Zhao
Rochester Institute of Technology
20 Lomb Memorial Dr, Rochester, NY 14623, USA
wjz@cs.rit.edu

ABSTRACT
We propose using "powered generalized min-max" (pGMM) hashed (linearized) via the "generalized consistent weighted sampling" (GCWS) for training (deep) neural networks (hence the name "GCWS-Net"). The pGMM and several related kernels were proposed in 2017 [27]. We demonstrate that pGMM hashed by GCWS provide a numerically stable scheme for applying power transformation on the original data, regardless of the magnitude of p and the data. Our experiments show that GCWSNet often improves the accuracy. It is also evident that GCWSNet converges substantially faster, reaching reasonable accuracy with merely one epoch of the training process. Empirical comparisons with (normalized) random Fourier features are provided. We also propose to reduce the model size of the neural networks become additions instead of multiplications because the input data become binary and highly sparse. This property is much desired because many applications, such as advertisement click-through rate (CTR) prediction models, or data streams (i.e., data seen only once), often train just one epoch. Another beneficial side effect is that the computations of the first layer of the neural networks become additions instead of multiplications because the input data become binary and highly sparse.

Empirical comparisons with (normalized) random Fourier features (NRFF) are provided. We also propose to reduce the model size of GCWSNet by count-sketch and develop the theory for analyzing the impact of using count-sketch on the accuracy of GCWS. Our analysis shows that an "8-bit" strategy should provide the good trade-off between accuracy and model size. There are other ways to take advantage of GCWS. For example, one can apply GCWS on the last layer to boost the accuracy of trained deep neural nets.¹

CCS CONCEPTS
• Computing methodologies → Machine learning; • Theory of computation → Theory and algorithms for application domains; • Information systems → Data management systems.

KEYWORDS
hashing, neural networks, pGMM kernel, consistent sampling

¹The open-source package https://github.com/pltrees/GCWSNet provides the implementation of GCWSNet with multiple deep learning platforms. In particular, the implementation in paddlepaddle was conducted while both authors were with the Cognitive Computing Lab, Baidu Research, 10900 NE 8th St. Bellevue, WA 98004, USA.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:
Ping Li and Weijie Zhao. 2022. GCWSNet: Generalized Consistent Weighted Sampling for Scalable and Accurate Training of Neural Networks. In Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM ’22), October 17–21, 2022, Atlanta, GA, USA. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3511808.3557332

1 INTRODUCTION
There has been a surge of interest in speeding up the training of large-scale learning algorithms. For example, [51] applied count-sketch type of randomized algorithms [6] to approximate large-scale linear classifiers. [33] applied (b-bit) minwise hashing [3, 4, 28, 30] to approximate the resemblance kernel in high-dimensional binary (0/1) data. In this paper, we propose using the "pGMM" kernel [27] and "generalized consistent weighted sampling" (GCWS) which approximates the pGMM kernel, for training neural networks.

The so-called "generalized min-max" (GMM) kernel was proposed in [26]. For defining the GMM kernel, the first step is a simple transformation on the original data. Consider, for example, the original data vector \( u_i \), \( i = 1 \ldots D \). The following transformation, depending on whether an entry \( u_j \) is positive or negative,

\[
\begin{align*}
\tilde{u}_{2i-1} &= u_i, & \tilde{u}_{2i} &= 0 & \text{if } u_i > 0 \\
\tilde{u}_{2i-1} &= 0, & \tilde{u}_{2i} &= u_i & \text{if } u_i \leq 0
\end{align*}
\]  

(1)

converts general data types to non-negative data only. For example, when \( D = 2 \) and \( u = [-3 \ 17] \), the transformed data vector becomes \( \tilde{u} = [0 \ 3 \ 17 \ 0] \). The GMM kernel is then defined as follows:

\[
\text{GMM}(u, v) = \frac{\sum_{i=1}^{2D}\min(\tilde{u}_i, \tilde{v}_i)}{\sum_{i=1}^{2D}\max(\tilde{u}_i, \tilde{v}_i)}. 
\]  

(2)

Note that, the GMM kernel in (2) still has no tuning parameter, unlike (e.g.,) the popular Gaussian (RBF) kernel. A simple strategy to introduce tuning parameters is the so-called "pGMM" kernel [27]:

\[
p\text{GMM}(u, v; p) = \frac{\sum_{i=1}^{2D}(\min(\tilde{u}_i, \tilde{v}_i))^p}{\sum_{i=1}^{2D}(\max(\tilde{u}_i, \tilde{v}_i))^p}, 
\]  

(3)

where \( p \in \mathbb{R} \) is a tuning parameter. Note that this is mathematically equivalent to first applying a power transformation on the data \((\tilde{u}, \tilde{v})\) before computing the GMM kernel. It will soon be clear that, combined with GCWS hashing, pGMM provides a numerically highly stable scheme for applying the power transformation.

1.1 Kernel SVM Experiments
While our focus is on training neural networks, we nevertheless provide a set of experimental studies on kernel SVMs for evaluating the pGMM kernel as in (3), in comparison with the linear kernel and the (best-tuned) RBF (Gaussian) kernel, as reported in Table 1.
We report test accuracies for the linear kernel, the best-tuned RBF kernel, the (tuning-free) GMM kernel, and the best-tuned pGMM kernel, at their individually-best SVM regularization C values. The results on linear kernels were obtained by LIBLINEAR [11] and the kernel SVM experiments were conducted using LIBSVM and pre-computed kernel matrices. The datasets are from the UCI repository, except for M-Noise1 and M-Image used by [22, 24, 27, 34] for testing deep learning algorithms and boosted trees. https://hunch.net/?p=1467

| Dataset      | # train, # test, # dim | Linear | RBF   | GMM   | pGMM (p) |
|--------------|-----------------|-------|-------|-------|----------|
| SEMG         | 1800, 1800, 2500 | 19.3  | 29.0  | 54.0  | 56.1 (2) |
| DailySports  | 4560, 4560, 3625 | 77.7  | 97.6  | 99.6  | 99.6 (0.6) |
| M-Noise1     | 10000, 4000, 784  | 60.3  | 66.8  | 71.4  | 85.2 (80) |
| M-Image      | 12000, 5000, 784  | 70.7  | 77.8  | 80.9  | 89.5 (50) |
| PAMAP101     | 188209, 188208, 51| 75.3  | —     | —     | —        |
| Covtype      | 290506, 290506, 54| 71.5  | —     | —     | —        |

For the SEMG dataset, the accuracies for the linear kernel and the (best-tuned) RBF kernel are very low, i.e., 19.3% and 29.0%, respectively. Perhaps surprisingly, the GMM kernel reaches 54% (with no tuning parameter) and the pGMM kernel achieves an accuracy of 56.1% at p = 2. For the M-Noise1 dataset, the best power parameter is p = 80 with an accuracy of 85.2%. Readers can check out the deep learning experiments reported in [22] and confirm that they are inferior to the pGMM kernel, for the M-Noise1 dataset.

It is not at all our intention to debate which type of classifiers work the best. We believe the performance highly depends on the datasets. Nevertheless, we hope that it is clear from Table 1 that the pGMM kernel is able to achieve comparable or better accuracy than the RBF kernel on a wide range of datasets. Note that, there are more than 10 datasets used in [22, 24], such as M-Noise2, ..., M-Noise6, etc which exhibit similar behaviors as M-Noise1.

1.2 Linearizing pGMM Kernel via Generalized Consistent Weighted Sampling

When using LIBSVM pre-computed kernel functionality, we find it is rather difficult if the number of training examples exceeds merely 30,000. It has been a well-known challenging task to scale up kernel learning for large datasets [2]. This has motivated many studies on approximating (linearizing) nonlinear kernels. In this paper, we adopt the “generalized consistent weighted sampling” (GCWS) to approximate the pGMM kernel, in the context of training neural networks for any tuning parameter p, as illustrated in Algorithm 1.

Algorithm 1 Generalized consistent weighted sampling (GCWS) for hashing the pGMM kernel.

Input: Data vector $u_i$ ($i = 1$ to $D$)

Generate vector $\tilde{u}$ in 2D-dim by (1).

for $i$ from 1 to 2D do

$\gamma^i \sim \text{Gamma}(2, 1)$, $c_i \sim \text{Gamma}(2, 1)$, $\beta_i \sim \text{Uniform}(0, 1)$

$t_1 \leftarrow \frac{\log(\tilde{u})}{\gamma^i} + \beta_i$, $a_i \leftarrow \log(c_i) - r_1(t_1 + 1 - \beta_i)$

end for

Output: $t^* \leftarrow \arg\min_i a_i$, $\tilde{t}^* \leftarrow t^*$

Algorithm 1 uses vector $u_i$ for the illustration. Given another vector $v$, we feed it to GCWS using the same set of random numbers: $r_1$, $c_1$, $\beta_1$. For differentiation, we name the hash samples as $(i^*_u, t^*_u)$ and $(i^*_v, t^*_v)$. We first present the basic probability result as a theorem.

Theorem 1.

$$P[(i^*_u, t^*_u) = (i^*_v, t^*_v)] = p\text{GMM}(u, v).$$

The proof of Theorem 1 directly follows from the basic theory of consistent weighted sampling (CWS) [18, 31, 37]. Although the original CWS algorithm is designed (and proved) only for non-negative data, we can see that the two transformations, i.e., converting general data types to non-negative data by (1) and applying the power on the converted data as in (3), are only pre-processing steps and the same proof for CWS will go through for GCWS.

Note that, in Algorithm 1, while the value of the output $t^*$ is upper bounded by 2D, the other integer output $t^*$ is unbounded. This makes it less convenient for the implementation. The next Theorem provides the basis for the (b-bit) implementation of GCWS.

Theorem 2. Assume that we can map $(i^*_u, t^*_u)$ uniformly to a space of $b$ bits denoted by $(i^*_u, t^*_u)_{b}$. Similarly, we have $(i^*_v, t^*_v)_{b}$. Then

$$P[(i^*_u, t^*_u)_{b} = (i^*_v, t^*_v)_{b}] = p\text{GMM}(u, v) + \frac{1}{2^b} \{1 - p\text{GMM}(u, v)\}.$$ (5)

1.3 Practical Implementation of GCWS

For each input data vector, we need to generate $k$ hash values, for example, $(i^*_{u,j}, t^*_{u,j}), j = 1$ to $k$, for data vector $u$. The mapping of $(i^*_{u,j}, t^*_{u,j})$ uniformly to $(i^*_{u,j}, t^*_{u,j})_{b}$ is actually not a trivial task, in part because $t^*$ is unbounded. Based on the intensive experimental results in [25] (for the original CWS algorithm), we will take advantage of following approximation:

$$P[i^*_u = i^*_v] \approx P[(i^*_u, t^*_u) = (i^*_v, t^*_v)] = p\text{GMM}(u, v).$$ (6)

and only keep the lowest $b$ bits of $i^*$ (unless we specify otherwise).

Suppose that, for data vector $u$, we obtain $k = 3$ hash values which, using only the lowest $b = 2$ bits, become $(3, 0, 1)$. We then concatenate their “one-hot” representation $[1 0 0 0 0 0 0 1 0 0 1 0]$ and feed it to subsequent tasks for classification, regression, or clustering. In other words, with $k$ hashes and $b$ bits, for each input vector we obtain a binary vector of length $2^b \times k$ with exactly $k$ 1’s.

The approximation in (6) (so called “0-bit” CWS) was purely an empirical (and mysterious) observation by [25]. Recently, [31] has partially resolved the mystery by developing a closely-related (but different) hashing method based on extremal processes.

1.4 History of Consistent Weighted Sampling

For binary (0/1) data, the pGMM kernel becomes the resemblance (Jaccard) similarity and GCWS is essentially equivalent to the celebrated minwise hashing algorithm [3, 4, 28, 30], with numerous applications [1, 5, 8–10, 13–15, 17, 19, 23, 33, 38–42, 45, 47–50, 54].

There exist strategies for generalizing the resemblance and minwise hashing for non-binary data. For example, CRS (Conditional Random Sampling) [28, 29] stores the smallest (bottom) $k$ non-zero entries for each row of the data matrix, after one (and only one) random permutation on the columns. CWS (consistent weighted sampling) is another strategy, originally for non-negative data.
The development of consistent weighted sampling (CWS) algorithm in its current form was due to [18, 37], as well as the earlier versions such as [16, 25] made the observation about the "0-bit" CWS, i.e., (6) and applied it to approximate large-scale kernel machines using linear algorithms. [26] generalized CWS to general data types which can have negative entries and demonstrated the considerable advantage of CWS over random Fourier features [43]. From the computational perspective, CWS and variants such as [31] are efficient in sparse data. For dense data, algorithms based on rejection sampling [7, 21, 36, 46] can be much more efficient than CWS. For relatively high-dimensional datasets, the method of "bin-wise CWS" (BCWS) [32] would always be recommended. Simply speaking, the basic idea of BCWS is to divide the data matrix into bins and then apply CWS (or sparse data) or rejection sampling (for dense data) in each bin. Finally, we should add that, despite the work [31], the mathematical explanation of the "0-bit" approximation (6), empirically observed by [25], remains an open problem.

1.5 Our Contributions

We develop GCWS (generalized CWS) for hashing the pGMM kernel and then apply GCWS for efficiently training neural networks. We name our procedure GCWSNet and show that GCWSNet can often achieve more accurate results compared with the standard neural networks. We also observe that the convergent speed of GCWSNet can be substantially faster than the standard neural networks. In fact, GCWS typically achieves a reasonable accuracy in (less than) one epoch. This property might be a huge advantage. For example, in many applications such as data streams, one of the 10 challenges in data mining [52], each sample vector is only seen once and hence only one epoch is allowed in the training process. In commercial search engines, when training click-through rate (CTR) deep learning models, typically only one epoch is used [12, 53] as the training size is on the petabyte scale and new observations (i.e., user clicks) keep arriving at a very fast rate. Additionally, there is a side benefit of GCWSNet, because the computations at the first layer become additions instead of multiplications.

2 GCWSNet

For each input data vector, we first generate \(k\)hash values \((i^*, r^*)\) using GCWS as in Algorithm 1. We adopt the "0-bit" approximation (6) and keep only the lowest \(b\) bits of \(i^*\). For example, suppose \(k = 3\), \(b = 2\) bits, and the hash values become \((3, 0, 1)\), which is the input data fed to neural networks. In short, with \(k\) hashes and \(b\) bits, for each input data vector we obtain a binary vector of length \(2^b \times k\) with exactly \(k\) 1's.

In the experiments, we adopted the Adam optimizer [20] and set the initial learning rate to be 0.001. We use "ReLU" for the activation function; other learning rates gave either similar or worse results in our experiments. The batch size was set to 32 for all datasets except Covtype for which we use 128 as the batch size. We report experiments for neural nets with no hidden units (denoted by "\(L = 1\)"), one layer of \(H\) hidden units (denoted by "\(L = 2\)"), and two layers (denoted by "\(L = 3\)") of hidden units (\(H\) units in the first hidden layer and \(H/2\) units in the second hidden layer). We have also experimented with more layers.

Figure 1 presents the results on the SEMG dataset using GCWS (with \(p = 1\)) along with the results on the original data (dashed black curves). This is a small dataset with 2,500 features. The dashed (black) curves for \(L = 1\) (i.e., neural nets with no hidden units) are consistent with the results in Table 1, i.e., an accuracy of \(< 20\%\) for linear classifier. Using one layer of \(H\) hidden units (i.e., \(L = 2\)) improves the accuracy noticeably (to \(25\%\)). Using two layers of hidden units (i.e., \(L = 3\)) further improves the accuracy only little.

For the SEMG dataset, GCWS is highly effective. We report the experiments for \(b \in \{1, 2, 4, 8\}\) and \(k \in \{64, 128, 256, 512\}\), on the SEMG dataset (see Table 1). We report the test accuracy for 5 epochs. "\(L = 1\)" means no hidden units, i.e., just logistic regression. "\(L = 2\)" means one hidden layer of \(H\) units. "\(L = 3\)" means two hidden layers with \(H\) units in the first hidden layer and \(H/2\) units in the second hidden layer. The results for the original data (dashed black curves) are much worse than GCWSNet, as expected from Table 1. One important observation is that GCWSNet converges fast, reaching a good accuracy after just one epoch of training.

Figure 1: GCWSNet with \(p = 1\), for \(b \in \{1, 2, 4, 8\}\) and \(k \in \{64, 128, 256, 512\}\), on the SEMG dataset (see Table 1). We report the test accuracy for 5 epochs. "\(L = 1\)" means no hidden units, i.e., just logistic regression. "\(L = 2\)" means one hidden layer of \(H\) units. "\(L = 3\)" means two hidden layers with \(H\) units in the first hidden layer and \(H/2\) units in the second hidden layer. The results for the original data (dashed black curves) are much worse than GCWSNet, as expected from Table 1. One important observation is that GCWSNet converges fast, reaching a good accuracy after just one epoch of training.
Figure 2: GCWSNet with \( p = 2 \) on the SEMG dataset. Note that for the original data (dashed black curves), we still report the original results without applying the power transformation. The results on original data with power transformation using \( p = 2 \) are actually not too much different from using \( p = 1 \).

Table 1 shows that, for the SEMG dataset, the pGMM kernel with \( p = 2 \) improves the classification accuracy. Thus, we also report the results of GCWSNet for \( p = 2 \) in Figure 2. Compared with Figure 1, the improvements are obvious. Figure 2 also again confirms that training just one epoch can already reach good accuracies.

Next we examine the experimental results on the Covtype dataset, in Figure 3 for 100 epochs. Again, we can see that GCWSNet converges substantially faster. To better view the convergence, we repeat the same plots in Figure 4 but for just 1 epoch. As the Covtype dataset has 290,506 training samples, we cannot compute the pGMM kernel directly. LIBLINEAR reports an accuracy of 71.5%, which is consistent with the results for \( L = 1 \) (i.e., no hidden layer).

The experiments on the Covtype dataset reveal a practically important issue, if applications only allow training for one epoch, even though training more epochs might lead to noticeably better accuracies. For example, consider the plots in the right-bottom corner in both Figure 3 and Figure 4, for \( b = 8, L = 2, H = 200 \). For the original data, if we stop after one epoch, the accuracy would drop from about 89% to 77%, corresponding to a 13.5% drop of accuracy. However, for GCWSNet with \( k = 64 \), if the training stops after one epoch, the accuracy would drop from 91% to just 84% (i.e., a 7.7% drop). This again confirms the benefits of GCWSNet in the scenario of one-epoch training.

Figure 5 reports the summary of the test accuracy for Covtype and just one epoch, to better illustrate the impact of \( b, H, k, \) and \( L \).

Figure 3: GCWSNet with \( p = 1 \), for \( b \in \{2, 4, 8\} \) and \( k \in \{64, 128, 256, 512\} \), on the Covtype dataset, for 100 epochs. The results for the original data are reported as dashed (black) curves, which are substantially worse than the results of GCWSNet. Again, GCWSNet converges much faster than training on the original data.

Figure 4: GCWSNet with \( p = 1 \), for \( b \in \{2, 4, 8\} \) and \( k \in \{64, 128, 256, 512\} \), on the Covtype dataset, for just one epoch, to illustrate that GCWSNet converges much faster.

Figure 5: Summary results of GCWSNet with \( p = 1 \) on the Covtype dataset, at the end of the first epoch.
Figure 6: GCWSNet with $p = 1$, for $b \in \{1, 2, 4\}$ and $k \in \{64, 128, 256, 512\}$, on the PAMAP101 dataset, for 100 epochs. Again, we can see that GCWSNet converges much faster than training on the original data.

Next, Figure 6 and Figure 7 present the experimental results on the PAMAP101 dataset, for We report the results for 100 epochs in Figure 6 and the results for just 1 epoch in Figure 7. Again, we can see that GCWSNet converges much faster and can reach a reasonable accuracy even with only one epoch of training.

Finally, we summarize the results on the DailySports dataset, the M-Noise1 dataset, and the M-Image dataset, in Figure 8, Figure 9, and Figure 10, respectively, for just 1 epoch. For DailySports, we let $p = 1$. For M-Noise1 and M-Image, we use $p = 80$ and $p = 50$, respectively, as suggested in Table 1.

3 COMBINING GCWS WITH COUNT-SKETCH

GCWS generates high-dimensional binary sparse inputs. With $k$ hashes and $b$ bits for encoding each hashed value, we obtain a vector of size $2^b \times k$ with exactly $k$ 1’s, for each original data vector. While the (online) training cost is mainly determined by $k$ the number of nonzero entries, the model size is still proportional to $2^b \times k$. This might be an issue for the GPU memory when both $k$ and $b$ are large. In this scenario, the well-known method of “count-sketch” might be helpful for reducing the model size.

Count-sketch [6] was originally developed for recovering sparse signals (e.g., “elephants” or “heavy hitters” in network/database terminology). [51] applied count-sketch as a dimension reduction tool. The key step is to independently and uniformly hash elements of the data vectors to buckets $\{1, 2, 3, ..., B\}$ and the hashed value is the weighted sum of the elements in the bucket, where the weights are generated from a random distribution which must be $\{-1, 1\}$ with equal probability (the reason will soon be clear). That is, we have $h(i) = j$ with probability $\frac{1}{B}$, where $j \in \{1, 2, ..., B\}$. For convenience, we introduce an indicator function:

$$I_{ij} = \begin{cases} 1 & \text{if } h(i) = j \\ 0 & \text{otherwise} \end{cases}$$

Consider two vectors $x, y \in \mathbb{R}^d$ and assume $d$ is divisible by $B$, without loss of generality. We also generate a random vector $r_i$, $i = 1$ to $d$, i.i.d., with the following property:

$$E(r_i) = 0, \quad E(r_i^2) = 1, \quad E(r_i^4) = 0, \quad E(r_i^4) = s$$ (7)
The above theorem implies that we must have $E(r^t_i) = s = 1$. There is only one such distribution, i.e., $\{-1, +1\}$ with equal probability.

Now we are ready to present our idea of combining GCWS with count-sketch. Recall that in the ideal scenario, we map the output of GCWS $(i^*, t^*)$ uniformly to a $b$-bit space of size $2^b$, denoted by $(i^*, t^*)_B$. For the original data vector $u$, we generate $k$ such (integer) hash values, denoted by $(i^*_{u,j}, t^*_{u,j})$, $j = 1$ to $k$. Then we concatenate the $k$ one-hot representations of $(i^*_{u,j}, t^*_{u,j})$, we obtain a binary vector of size $d = 2^b \times k$ with exactly $k$ 1’s. Similarly, we have $(i^*_{w,j}, t^*_{w,j})$, $j = 1$ to $k$ and the corresponding binary vector of length $2^b \times k$. Once we have the sparse binary vectors, we can apply count-sketch with $B$ bins. For convenience, we denote

$$B = (2^b \times k)/m.$$  \hfill (11)

That is, $m$ represents the dimension reduction factor, and $m = 1$ means no reduction. We can then generate the $(B$-bin) count-sketch samples, $z$ and $w$, as in (8). We present the theoretical results on the inner product $<z, w>$ as the next theorem, in a way that its proof should be self-explanatory.

**Theorem 4.** Consider original data vector $u$ and $v$. Denote $P_b = P \{ (i^*_{v,j}, t^*_{v,j}) \in (i^*_{u,j}, t^*_{u,j}) \}$. Vector $z$ of size $B$ is the $(B$-bin) count-sketch samples generated from the concatenated binary vector from the one-hot representation of $(i^*_{u,j}, t^*_{u,j})$. Similarly, $w$ of size $B$ is the count-sketch samples for $v$. Also, denote

$$a = \sum_{j=1}^{k} (i^*_{u,j}, t^*_{u,j}) = (i^*_{w,j}, t^*_{w,j}).$$  \hfill (12)

Conditioning on the GCWS output $(i^*_{u,j}, t^*_{u,j})$, $(i^*_{w,j}, t^*_{w,j})$ and using Theorem 3 with $s = 1$, we have

$$E(<z, w > | GCWS) = a,$$  \hfill (13)

$$Var(<z, w > | GCWS) = \frac{1}{B} [k^2 + a^2 - 2a].$$  \hfill (14)

Note that $a$ is binomial Bin$(k, P_b)$, i.e., $E(a) = kP_b$, $Var(a) = kP_b(1-P_b)$. Therefore, unconditionally, we have

$$E(<z, w >) = kP_b,$$  \hfill (15)

$$Var(<z, w >) = \frac{1}{B} [k^2 + k^2P^2_b - kP^2_b - kP_b] + kP_b(1-P_b).$$  \hfill (16)

Define the estimator $\hat{P}_b$ as

$$\hat{P}_b = \frac{<z, w >}{k}.$$  \hfill (17)

Then

$$E(\hat{P}_b) = P_b,$$  \hfill (18)

$$Var(\hat{P}_b) = \frac{P_b(1-P_b)}{k} + \frac{1}{B} \left[1 + P^2_b - P^2_b/k - P_b/k \right].$$  \hfill (19)

In the above Theorem, the variance $Var(\hat{P}_b)$ (19) in Theorem 4 has two parts. The first term $\frac{P_b(1-P_b)}{k}$ is the usual variance without using count-sketch. The second term $\frac{1}{B} \left[1 + P^2_b - P^2_b/k - P_b/k \right]$ represents the additional variance due to count-sketch. The hope is that the additional variance would not be too large, compared with $P_b(1-P_b)$. Let $B = (2^b \times k)/m$, then the second term can be written as

$$\frac{1}{k \cdot 2^b} \left[1 + P^2_b - P^2_b/k - P_b/k \right].$$

To assess the impact of count-sketch, we need to compare $P_b(1-P_b)$ with $\frac{m}{2^b} \left[1 + P^2_b - P^2_b/k - P_b/k \right]$, which is approximately $\frac{m}{2^b} \left[1 + P^2_b \right]$, as $k$ has to be sufficiently large. Recall from Theorem 2 $P_b = J + \frac{1}{2^b}(1 - J)$, where $f =$
\(pGMM(u, v; p) = \frac{e^{\frac{p-1}{2}}}{\sum_{i=1}^{m} (\min(\hat{u}_i, \hat{v}_i))^{p}}\). We can hence resort to the following "ratio" to assess the impact of count-sketch on the variance:

\[ R = R(b, J, m) = \frac{m}{2^b} \frac{1 + p}{P_b} \left( \frac{1}{2^b J} \right) \text{ where } P_b = J + \frac{1}{2^b} (1 - J). \tag{20} \]

Figure 11: The ratio (20) \(R = \frac{m}{2^b} \frac{1 + p}{P_b} \left( \frac{1}{2^b J} \right)\), where \(P_b = J + \frac{1}{2^b} (1 - J)\) and \(J\) the pGMM similarity of interest, for \(b \in \{8, 12, 16\}\) and a series of \(m\) values. Ideally, we would like to see large \(m\) and small ratio values.

As shown in Figure 11, when \(b = 16\), the ratio is small even for \(m = 1000\). However, when \(b = 8\), the ratio is not too small after \(m > 10\); and hence we should only expect a saving by an order magnitude if \(b = 8\) is used. The choice of \(b\) has, to a good extent, to do with \(D\), the original data dimension. When the data are extremely high-dimensional, say \(D = 2^{30}\), we expect an excellent storage savings can be achieved by using a large \(b\) and a large \(m\).

For practical consideration, we consider two strategies for choosing \(m\), as illustrated in Figure 12. The first strategy (left panel) is "always using half of the bits", that is, let \(m = 2^b/2\). This method is probably a bit conservative for \(b \geq 16\). The second strategy (right panel) is "always using 8 bits" which corresponds to almost like a single curve, because \(R = \frac{1}{2^b} \frac{1 + p}{P_b} \left( \frac{1}{2^b J} \right)\) in this case and \(P_b = J + \frac{1}{2^b} (1 - J)\) when \(b \geq 8\).

We believe the "always using 8 bits" strategy might be a good practice. In implementation, it is convenient to use 8 bits (i.e., one byte). Even when we really just need 5 or 6 bits, we might as well simply use one byte to avoid the trouble of performing bits-packing.

We conclude this section by providing an experimental study on M-Noise1 and M-Image and summarize the results in Figure 13. For both datasets, when \(m \ll 16\), we do not see an obvious drop of accuracy, compared with \(m = 1\) (i.e., without using count-sketch). This confirms the effectiveness of our proposed procedure by combining GCWSNet with count-sketch.

Figure 12: The ratio (20) for two strategies of choosing \(m\). Left panel: we let \(m = 2^b/2\) (i.e., always using half of the bits). Right panel: for \(b \geq 8\) we let \(m = 2^b-8\) (i.e., always use 8 bits).

Figure 13: GCWS combined with count-sketch for reducing the model size by a factor of \(m\), where \(m = (2^b \times k)/B\) and \(B\) is the number of bins in the count-sketch step. \(m = 1\) means there is no reduction. We can see that at least with \(m = 8\) or \(m = 16\), there is no obvious loss of accuracy.

4 ROBUST POWER TRANSFORMATION

In Algorithm 1, this step (on nonzero entries)

\[ t_i \leftarrow \left[ p \log \frac{u_i}{r_i} + \beta_i \right] \tag{21} \]

suggests that GCWS can be viewed as a robust power transformation on the data. It should be clear that GCWS is not simply taking the log-transformation on the original data. We compare three different strategies for data preprocessing. (i) GCWS; (ii) feeding the power transformed data (e.g., \(u^p\)) to neural nets; (iii) feeding the log-power transformed data (e.g., \(\log u\)) to neural nets.

Naively applying power transformation on the original data might encounter practical problems. For example, when data entries contain large values such as "1533" or "396", applying a powerful transformation such as \(p = 80\) might lead to overflow problems and other issues such as loss of accuracy during computations. Using the log-power transformation, i.e., \(p \times \log u\), should alleviate many problems but new issues might arise. For example, when data entries \(u\) are small. How to handle zeros is another headache with log-power transformation.

Here, we provide an experimental study on the UCI Pendigits dataset, which contains positive integers features (and many zeros). The first a few entries of the training dataset are "8 1:47 2:100 3:27 4:81 5:57 6:76 10:26 11:56" (in LIBSVM format). As shown in Figure 14, after \(p \geq 15\), directly applying the power transformation (i.e., \(u^p\)) makes the training fail. The log-transformation (i.e., \(\log u\)) seems to be quite robust in this dataset, because we use a trick by letting \(p \log 0 = 0\), which seems to work well for this dataset. On the other hand, GCWS with any \(p\) produces reasonable predictions, although it appears that \(p \in [0.5, 5]\) (a wide range) is the optimal range for GCWSNet. The experiments on Pendigits confirm the robustness of GCWSNet with any \(p\). Additionally, the experiments once again verify that GCWSNet converges really fast and achieves a reasonable accuracy as early as in 0.1 epoch.
which, initially, was purely an empirical observation [25]. Char-
acterizing this approximation for CWS remains an open problem.

To conclude this section, we should emphasize that we do not
claim that we have fully solved the data preprocessing problem,
which is a highly crucial task for practical applications. We simply
introduce GCWSNet with one single tuning parameter which hap-
pens to be quite robust, in comparison with obvious (and commonly
used) alternatives for power transformations.

5 USING BITS FROM $t^*$

In Figure 16, we report the experiments for using 1 or 2 bits from $t^*$,
which have not been used in all previously presented experiments.

Recall the approximation in (6)

$$P[i_i^u = i_i^v] \approx P[(i_i^u, t^*) = (i_i^v, t^*_i)] = pGMM(u, v),$$

which, initially, was purely an empirical observation [25]. Char-
acterizing this approximation for CWS remains an open problem.

Figure 16 provides a validation study on M-Noise and M-Image,
by adding the lowest 1 bit (dashed curve) or 2 bits (dashed dot curves)
of $t^*$. We can see that adding 1 bit of $t^*$ indeed helps slightly when
we only use $b = 1$ or $b = 2$ bits for $i^*$. Using 2 bits for $t^*$ does not
lead further improvements. When we already use $b = 4$ or $b = 8$

bits to encode $i^*$, then adding the information for 1 bit from $t^*$ does
not help in a noticeable manner.

We expect that practitioners would anyway use a sufficient num-
ber (e.g., 8) of bits for $t^*$. We therefore do not expect that using bits
from $t^*$ would help. Nevertheless, if it is affordable, we would
suggest using the lowest 1 bit of $t^*$ in addition to using $b$ bits for $i^*$.

6 COMPARISON WITH NORMALIZED RANDOM FOURIER FEATURES (NRFF)

The method of random Fourier features (RFF) [35, 43, 44] is popular
for approximating the RBF (Gaussian) kernel. For convenience, the

Figure 14: GCWS with a wide range of $p$ values, on the UCI
Pendigits dataset, for $b = 8$ and $k \in \{64, 512\}$. We can see
that using $p \in [0.5, 5]$ produces the best results. Also, the
results of GCWSNet are still reasonable even with very large
or very small $p$ values. In comparison, we apply the power
transformation (i.e., $u^p$) and log-power transformation (i.e.,
$p \log u$, treating $u \log 0 = 0$) on the original data and feed the
transformed data directly to neural nets. Their results are
not as accurate compared with GCWSNet. Also, the power
transformation encountered numerical issues with $p \geq 15$.

Figure 15: GCWS with different $p$ values, on the M-Noise1
dataset, for $b = 8$ and $k \in \{64, 512\}$. We can see that using $p \in
[60, 150]$ produces the best results. The results of GCWSNet
are still reasonable even with $p = 1$. Using the log-power
transformation produces bad results except for $p$ close to 1.

Figure 16: GCWSNet for M-Noise (with $p = 80$) and M-Image
(with $p = 50$) by using 0 bit (solid), 1 bit (dashed), and 2 bits
dashed dot) for $t^*$ in the output of GCWS hashing ($i^*, t^*$)
input data vectors are normalized to have unit \( l_2 \) norms, i.e., \( \| u \| = \| o \| = 1 \). The RBF (Gaussian) kernel is then \( \text{RBF}(u, v, y) = e^{-\gamma(1-\rho)} \), where \( \rho = <u, v> \) is the cosine and \( \gamma > 0 \) is a tuning parameter. We sample \( w \sim \text{uniform}(0,2\pi) \), \( r_i \sim N(0,1) \) i.i.d., and denote \( x = \sum_{i,j} w_ir_{ij} \), \( y = \sum_{j} D_{ij} r_{ij} \). It is clear that \( x \sim N(0,1) \) and \( y \sim N(0,1) \). The procedure is repeated \( k \) times to generate \( k \) RFF samples for each data vector. [26] advocated the so-called “normalized RFF” (NRFF) by normalizing the RFF’s to unit \( L_2 \) norm, with theoretical support, because the theoretical variance after normalization becomes much smaller.

Figure 17 provides an experimental study to compare NRFF with GCWS on two datasets: SEMG and DailySports and \( b \in \{1, 8\} \). We use much larger \( k \) values for NRFF, as large as 8192.

Figure 17: NRFF (dashed curves) versus GCWS (solid curves), for two datasets: SEMG and DailySports and \( b \in \{1, 8\} \). We use much larger \( k \) values for NRFF, as large as 8192.

8 CONCLUDING REMARKS

In this paper, we adopt the pGMM kernel developed in 2017 [27] with a single tuning parameter \( p \) and use GCWS (generalized consistent weighted sampling) to generate hash values, producing sparse binary data vectors of size \( 2^b \times k \), where \( k \) is the number of hash samples and \( b \) is the number of bits to store each hash value. These binary vectors are fed to neural networks for subsequent tasks such as regression or classification. Our extensive experiments show that GCWS converges fast and typically reaches a reasonable accuracy at (less than) one epoch of training. This property of GCWS could be highly beneficial in practice because many important applications such as data streams or CTR predictions in commercial search engines typically train the models only with one epoch.

GCWS with a tuning parameter \( p \) provides a beneficial and robust power transformation on the original data. By adding this tuning parameter, the performance can often be improved, in some cases considerably so. Inside GCWS, this parameter acts on the (nonzero) data entry as \( p \log u \). The nature of GCWS is that the coordinates of entries with larger values have a higher chance to be picked as output (in a probabilistic manner). In other words, both the absolute values and relative orders matter, and the power transformation does impact the performance. Experiments show that GCWSNet with a tuning parameter \( p \) produces more accurate results compared with two obvious strategies: 1) feeding power-transformed data \( u^p \) directly to neural nets; 2) feeding log-power-transformed data \( p \log u \), with zeros handled separately) directly to neural nets.

GCWSNet can be combined with count-sketch to reduce the model size of GCWSNet, which is proportional to \( 2^b \times k \) and can be pretty large if \( b \) is large such as 16 or 24. Our theoretical analysis for the impact of count-sketch on the estimation variance provides the explanation for the effectiveness of count-sketch demonstrated in the experiments. We recommend the “8-bit” practice in that we always use \( 2^8 \) bins for count-sketch if GCWS uses \( b \geq 8 \). With this strategy, even when \( k = 2^{10} = 1024 \), the model size is only proportional to \( 2^8 \times 2^{10} = 2^{18} \), which is not a large number.

There are other ways to take advantage of GCWS. For example, one can always apply GCWS on the embeddings of trained neural nets to hopefully boost the performance of existing network models. We also hope the comparison of GCWS with the popular random Fourier features would promote interest in future research.
[48] Christopher Thomas and Adriana Kovashka. Preserving semantic neighborhoods for robust cross-modal retrieval. In *Proceedings of the 16th European Conference on Computer Vision (ECCV), Part XVIII*, pages 317–335, Glasgow, UK, 2020.

[49] Kateryna Tymoshenko and Alessandro Moschitti. Cross-pair text representations for answer sentence selection. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2162–2173, Brussels, Belgium, 2018.

[50] Tanguy Urvoy, Emmanuel Chauveau, Pascal Filoche, and Thomas Lavergne. Tracking web spam with html style similarities. *ACM Trans. Web*, 2(1):1–28, 2008.

[51] Kilian Q. Weinberger, Anirban Dasgupta, John Langford, Alexander J. Smola, and Jodi Attenberg. Feature hashing for large scale multikernel learning. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, pages 1113–1120, Montreal, Canada, 2009.

[52] Qiang Yang and Xindong Wu. 10 challenging problems in data mining research. *International Journal of Information Technology & Decision Making*, 5(04):597–604, 2006.

[53] Weijie Zhao, Deping Xie, Ronglai Jia, Yulei Qian, Ruiquan Ding, Mingming Sun, and Ping Li. Distributed hierarchical GPU parameter server for massive scale deep learning ads systems. In *Proceedings of Machine Learning and Systems 2020 (MLSys)*, Austin, TX, 2020.

[54] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. JOSIE: overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD)*, pages 847–864, Amsterdam, The Netherlands, 2019.