Gated Recurrent Unit の簡略化と学習型 Bloom Filter への影響

大西雄真 [1]　西田拳 [1]　林克彦 [2]　上垣外英剛 [3]
[1] 北海道大学　[2] 東京大学　[3] 奈良先端科学技術大学院大学
onishi.kazuma.l5@elms.hokudai.ac.jp　kenmankita@eis.hokudai.ac.jp
katsuhiko-hayashi@g.ecc.u-tokyo.ac.jp　kamigaito.h@is.naist.jp

概要

機械学習と Bloom Filter (BF) を組み合わせることで，BF のメモリ使用量や計算効率を向上させる Learned Bloom Filter (LBF) が注目を集めている．BF としての特性上，LBF で採用する機械学習モデルは軽量かつ計算効率に優れていることが求められるが，系列データに対する LBF において，どのような機械学習モデルを採用すれば良いかはまだ十分に議論されていない．本研究では，系列データに対する LBF の機械学習モデルとして，Gated Recurrent Unit (GRU) を考える．また，その構造の簡略化も検討し，それが LBF の性能に与える影響を調査する．

1　はじめに

Web サービスにおけるフィッシング URL の検出や，大規模なクラウドストレージシステムでの重複データの削除など，大規模なデータの管理においては，ある要素がデータセットに存在するか否かを高速かつ省メモリで判定する技術が重要となる．Bloom Filter (BF) [1] は軽量かつ高速なメンバシップ判定を可能にする確率的データ構造であり，ビット配列と複数のハッシュ関数を用いて要素の存在を効率的に判定する．BF の適用範囲は広く，自然言語処理分野では，スパム URL フィルタ [2, 3] から，*n*-gram 言語モデルの効率化 [4] や語彙サイズの圧縮 [5] にも応用されている．また，ケモインフォマティクス分野における化合物の毒性判別 [6, 7] など系列データを扱う他の分野でも BF の活用が検討され始めている．

一方で，「存在しない」要素を「存在する」と判定する偽陽性 (FP; False Positive) を抑制するためには，ビット配列のサイズを拡張する必要がある．そのため，BF はデータ量の増加に伴いメモリ使用量が大きくなるという課題を抱えており，スマートフォンなどの低リソース環境での応用において問題となる．この課題に対処する手法として BF に機械学習モデルを組み合わせた Learned Bloom Filter (LBF) [8] が提案されている．LBF では，機械学習モデルによる予測を通じて，不要なメモリの使用を削減する．これにより，偽陽性の発生を抑えながら高速かつ，より軽量なメンバシップ判定を可能にしている．BF としての特性上，LBF で採用する機械学習モデルは軽量かつ計算効率に優れていることが求められるが，自然言語処理やケモインフォマティクスのような分野で扱う系列データに対して，どのような機械学習モデルを採用すれば良いかこれまでほとんど議論されていない．

本稿では，系列データとの親和性の高い再帰型ニューラルネットワーク (RNN) モデルを検討し，その一種である Gated Recurrent Unit (GRU) [9] に着目する．GRU は RNN における勾配消失の問題を緩和するため，ゲート構造を採用しているが，RNN の一種で同様にゲート構造を持つ Long Short-term Memory [10] よりもゲート数が少なく，パラメータ数も少ない．また，近年では，LSTM や GRU の計算を並列化するため，それらの構造を簡略化する研究が注目を集めている [11, 12, 13]．本稿でも，GRU の簡略化を行い，LBF における機械学習モデルの軽量化および計算効率の向上を検討する．また，GRU の簡略化が系列データに対する LBF の性能にどのような影響を与えるか調査する．

2　予備知識

2.1　Bloom Filter

Bloom filter (BF) [1] とは Bloom によって提案されたデータセットに特定の要素が存在するか否かを効

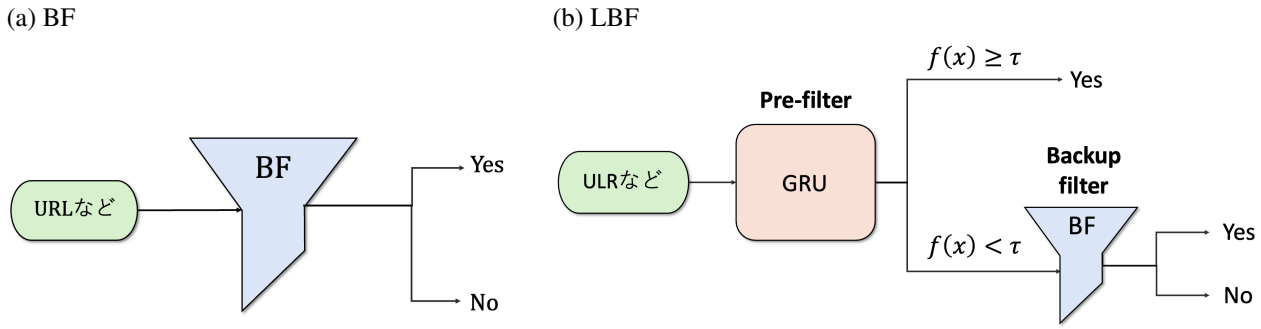


図 1　(a) BF 及び (b) LBF の構造の比較.

率的に判定するための確率的データ構造である．その主な特徴として，要素が「存在しない」場合は確実に判定ができる一方で，要素が「存在する」と判定された場合には偽陽性 (FP; False Positive) が発生する可能性がある．偽陰性 (FN; False Negative) の可能性がない点と，計算効率が優れていることから BF は「要素の存在判定」が求められる大規模なデータセットの管理で利用されている．

2.2　Bloom Filter の構築及び判定

BF は k 個の独立したハッシュ関数 $h_1, h_2, \ldots, h_k$ と長さ m の 0 に初期化されたビット配列を用いて構成される．要素 x を挿入する際には，各ハッシュ関数 h_i によって計算された位置 $h_i(x)$ に対応するビットをすべて 1 に設定する．この処理はすべての k 個のハッシュ関数で行われる．要素 y の存在を検索するときには，挿入操作と同様に各ハッシュ関数を用いて位置 $h_i(y)$ を計算し，すべての位置に対応するビットが 1 であれば要素 y は「存在する」と判定する．一方で，少なくとも一つのビットが 0 であれば要素 y は「存在しない」と判定される．BF の性能は，ビット配列の長さ m とハッシュ関数の数 k に大きく依存する．これらのパラメータは許容する偽陽性率 p と挿入する要素数 n を基に決定される (式 (1)，式 (2))．

$$m = -\frac{n \ln p}{(\ln 2)^2} \tag{1}$$

$$k = \frac{m}{n} \ln 2. \tag{2}$$

ハッシュ関数は，CRC32 [14] や SHA-256 [15] が広く採用されている．

2.3　Learned Bloom Filter

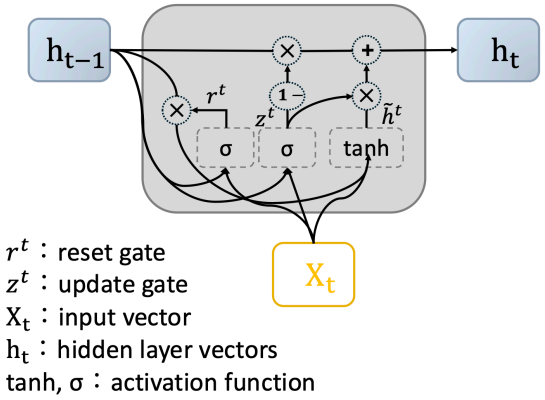
Learned Bloom Filter (LBF) [8] は，対象要素の存在判定を二値分類問題として扱い，入力された要素がデータ集合に含まれるかどうかを予測することが可能である．図 1 は単一の BF と LBF のアーキテクチャを示す．LBF は，機械学習モデルと BF の二段階のフィルタリングを行い，従来の BF に比べて効率的かつ正確な存在判定を実現する．まず一段階目のフィルタの機械学習モデルでは，対象要素の存在確率 $f(x)$ が出力される．その後設定した閾値 τ と比較を行い，$f(x)$ が τ より大きな値の場合，対象要素は「存在する」と判定される．一方，$f(x)$ が τ より小さな値の場合，二段階目のフィルタである BF で存在判定を行う．これにより，BF のみに比べてメモリ効率の向上が期待される．

3　GRU の簡略化

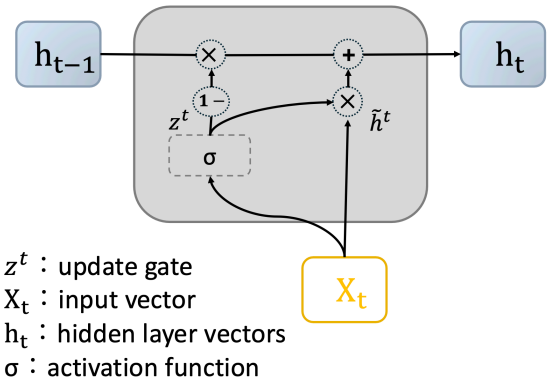
系列データを扱う LBF の機械学習モデル (一段階目のフィルタ) としては，系列データへの親和性の高さから，Gated Recurrent Unit (GRU) [9] を用いることが考えられる．GRU は，再起型ニューラルネットワーク (RNN) [16] の拡張である．RNN は過去の状態を隠れ状態として保持し，それを利用しながら現在の入力を処理する．GRU では，RNN が抱える勾配消失の問題を緩和するため Long Short-Term Memory (LSTM) [10] と同様にゲート構造を導入しており，以下の式 (3) で隠れ状態 $\mathbf{h}_t$ を更新する．

$$\begin{aligned}
\mathbf{z}_t &= \sigma\left(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1} + \mathbf{b}_z\right), \\
\mathbf{r}_t &= \sigma\left(W_r \mathbf{x}_t + U_r \mathbf{h}_{t-1} + \mathbf{b}_r\right), \\
\tilde{\mathbf{h}}_t &= \tanh\left(W_h \mathbf{x}_t + U_h (\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h\right), \\
\mathbf{h}_t &= \left(1 - \mathbf{z}_t\right) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t,
\end{aligned} \tag{3}$$

ここで，$\mathbf{x}_t$ は時刻 t における入力ベクトル，$\tilde{\mathbf{h}}_t$ は候補隠れ状態を表す．$\mathbf{z}_t$ は現在の隠れ状態 $\mathbf{h}_t$ を更新する割合を制御する更新ゲート (Update gate)，$\mathbf{r}_t$ は過去の隠れ状態 $\mathbf{h}_{t-1}$ をどの程度リセットするかを制御するリセットゲート (Reset gate) となる．また，$\sigma(\cdot)$ はシグモイド関数，$\tanh(\cdot)$ は双曲線正接関



(a) GRU



(b) minGRU

図 2　(a) GRU と (b) minGRU の内部構造比較.

数，⊙ は要素ごとの積 (Hadamard 積) を表す．W_i，U_i ($i \in \{z, r, h\}$) はそれぞれ，入力ベクトル $\mathbf{x}_t$，過去の隠れ状態 h_{t-1} に関連する重み行列を表し，$\mathbf{b}_i$ はバイアス項を示す．

これらのゲート構造により，勾配消失が緩和され，GRU は，系列データの長期的な依存関係を効果的に捉えることが可能である．一方で，GRU では各時刻 t において，ゲート及び隠れ状態の更新のすべてが前時刻の隠れ状態 $\mathbf{h}_{t-1}$ に依存するため，計算全体を逐次的に行う必要があり，時間方向での並列化が困難である．この逐次計算や複数のゲート構造は計算コストとメモリ使用量を増大させる要因となるため，大規模なデータセットを扱う際には，更なる効率化を目指した簡略化が求められる．

そこで本研究では，LBF の機械学習モデルに対して GRU を簡略化した minGRU [12] を適用した．

図 2 は GRU と minGRU の内部構造を示している．minGRU は GRU のゲート構造を大幅に簡略化しており，以下の式 (4) で動作する．

$$\begin{aligned} \mathbf{z}_t &= \sigma\left(W_z \mathbf{x}_t + \mathbf{b}_z\right), \\ \tilde{\mathbf{h}}_t &= W_h \mathbf{x}_t + \mathbf{b}_h, \\ \mathbf{h}_t &= (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t, \end{aligned} \tag{4}$$

minGRU では，リセットゲートが省略されている．それに伴い，リセットゲートに関連した重みパラメータの削減に加えて，候補隠れ状態 $\tilde{\mathbf{h}}_t$ の計算で非線形変換が不要になり，GRU に比べてより，軽量かつ計算効率が高くなっている．また，候補隠れ状態が過去の状態に依存しなくなったため，各時刻 t の候補隠れ状態は時間方向で並列に計算することが可能である．そのため，大規模なデータの処理や，リアルタイムでの処理が求められる環境において，実用性が高いと考えられる．

4　実験

4.1　データセット

本研究では，提案手法の性能を検証するため，有害・無害の二値ラベルが付与された URL を含むデータセット [17] を使用した．データセットには無害なサイトの URL に加えて，フィッシング URL やマルウェア URL などの有害な URL がラベル付きで格納されている．訓練データのサンプル数は 506,006 件，評価データのサンプル数は 126,502 件であり，正例（有害）データ数は 316,254 件，負例（無害）データ数は 316,254 件という内訳となっている．系列データの平均長は 23.2 である．

4.2　評価指標

評価指標として，偽陽性率 (FPR:False Positive Rate) とメモリ使用量に加えて存在判定にかかる処理速度を用いた．FPR は BF の判定精度を評価するための重要な指標である (式 5)．

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{5}$$

ここで，FP は偽陽性の個数，TN は真陰性の個数を示す．

4.3　実験設定

GRU，minGRU ともに入力の埋め込み次元を 16，隠れ層サイズを 64 に固定して判定性能を評価した．実験では，閾値 τ と BF の偽陽性率 p を調整することで，LBF モデルの性能を調整した．また，GRU，

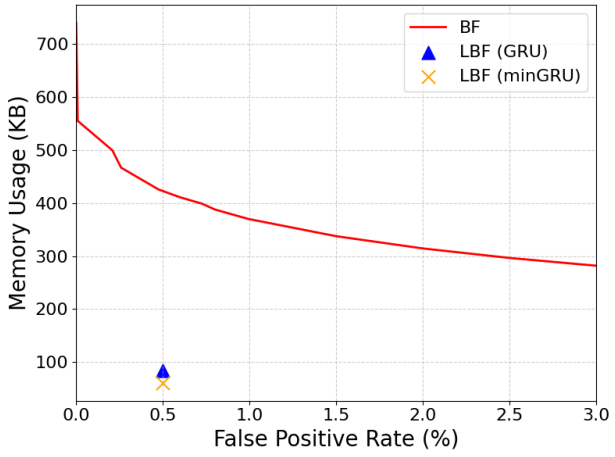


図 3　BF と LBF (GRU)，LBF (minGRU) のメモリ使用量の比較．縦軸がメモリ使用量，横軸が FPR (%) を示す．

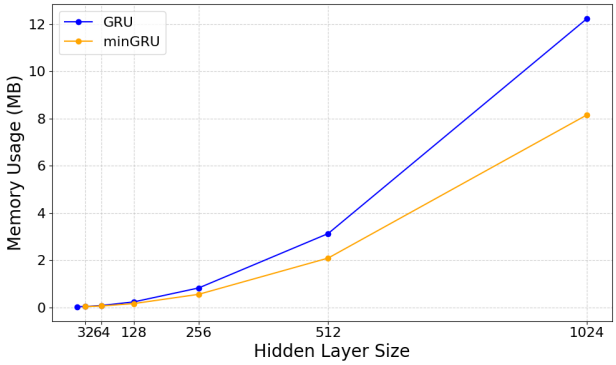


図 4　GRU と minGRU のメモリ使用量の比較．縦軸はメモリ使用量，横軸は隠れ層サイズを示す．

minGRU のハイパーパラメータによる変化を検証するため，埋め込み次元を 16 に固定し，隠れ層サイズを $\{16, 32, 64, 128, 256, 512, 1024\}$ と設定した場合のメモリ使用量の変化を比較した．加えて，サンプル数を増加させた場合の判定処理にかかる時間の変化についても検証した．

4.4　実験結果

図 3 は FPR とメモリ使用量の関係を示したグラフである．BF では，FPR が 0.5 を達成するのに **425 キロバイト**必要としたのに対し，GRU を使用した LBF では **81 キロバイト**，minGRU を使用した LBF では **61 キロバイト**であった．さらに，図 4 は隠れ層サイズを操作したときの GRU と minGRU のメモリ使用量の比較を示し，minGRU が GRU と比べて一貫してより軽量であることが確認できた．また，図 5 は判定処理にかかる時間の比較を示しており，簡略化したモデルがより高速に要素の存在判定を行えることがわかる．

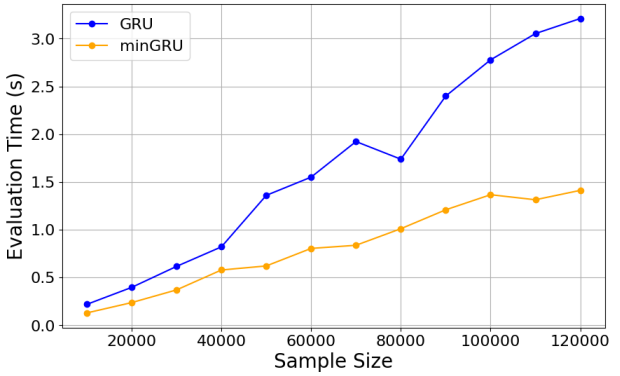


図 5　LBF (GRU) と LBF (minGRU) の処理速度の比較．縦軸は判定処理にかかった時間 (s)，横軸は入力サンプル数を示す．

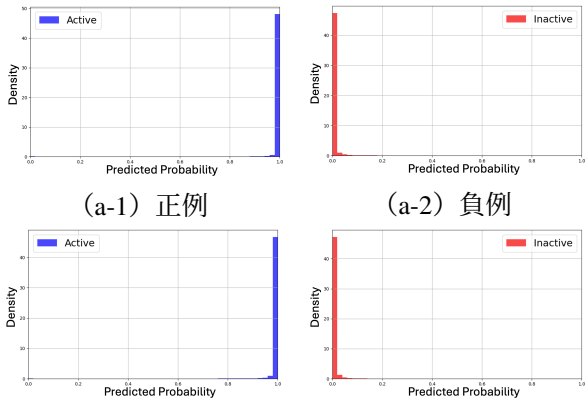


図 6　(a) GRU の予測スコア分布．(b) minGRU の予測スコア分布．

図 6 は機械学習モデルの予測スコアの分布を示している．どちらのモデルも高い精度で正例と負例を予測できており，GRU に比べて軽量な minGRU でも予測性能を維持できていることが確認できた．

これらの結果から，提案手法は従来の GRU を使用した LBF に比べて判定性能を維持しつつ，計算効率及びメモリ効率が高いことがわかる．

5　まとめ

本研究では，系列データを扱う LBF の機械学習モデルとして GRU に着目し，その簡略化を行なった．モデルは GRU のゲート構造を簡略化したものであり，学習時の処理の一部を並列化可能にしたのに加えて，パラメータ数が削減されているため，軽量化と計算効率の向上を実現している．実験では，GRU を利用した LBF と簡略化したモデルの比較を行い，提案手法が判定精度を維持しながら判定速度の向上に加えてメモリ使用量の削減を実現していることが確認できた．

参考文献

[1] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. **Commun. ACM**, Vol. 13, No. 7, p. 422–426, jul 1970.

[2] Jeff Yan and Pook Leong Cho. Enhancing collaborative spam detection with bloom filters. In **2006 22nd Annual Computer Security Applications Conference (ACSAC'06)**, pp. 414–428, 2006.

[3] Ripon Patgiri, Anupam Biswas, and Sabuzima Nayak. deepbf: Malicious url detection using learned bloom filter and evolutionary deep learning. **Computer Communications**, Vol. 200, pp. 30–41, 2023.

[4] David Talbot and Miles Osborne. Smoothed Bloom filter language models: Tera-scale LMs on the cheap. In Jason Eisner, editor, **Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)**, pp. 468–476, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[5] John Anderson, Qingqing Huang, Walid Krichene, Steffen Rendle, and Li Zhang. Superbloom: Bloom filter meets transformer, 2020.

[6] Jorge Medina and Andrew D White. Bloom filters for molecules, 2023.

[7] 西田拳, 林克彦, 上垣外英剛, 進藤裕之. Learned bloom filter を用いた化合物検索の効率化. 日本化学会 第 47 回ケモインフォマティクス討論会, 12 2024.

[8] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In **Proceedings of the 2018 International Conference on Management of Data**, SIGMOD '18, p. 489–504, New York, NY, USA, 2018. Association for Computing Machinery.

[9] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[10] S Hochreiter. Long short-term memory. **Neural Computation MIT-Press**, 1997.

[11] Eric Martin and Chris Cundy. Parallelizing linear recurrent neural nets over sequence length, 2018.

[12] Leo Feng, Frederick Tung, Mohamed Osama Ahmed, Yoshua Bengio, and Hossein Hajimirsadeghi. Were rnns all we needed?, 2024.

[13] Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory, 2024.

[14] T.-B. Pei and C. Zukowski. High-speed parallel crc circuits in vlsi. **IEEE Transactions on Communications**, Vol. 40, No. 4, pp. 653–657, 1992.

[15] Alok K. Kasgar, Jitendra Agrawal, and Satntosh Shahu. New modified 256-bit md5 algorithm with sha compression function. **International Journal of Computer Applications**, Vol. 42, pp. 47–51, 2012.

[16] I Sutskever. Sequence to sequence learning with neural networks. **arXiv preprint arXiv:1409.3215**, 2014.

[17] Benign and malicious urls. https://www.kaggle.com/datasetURL. Accessed: 2025-01-08.