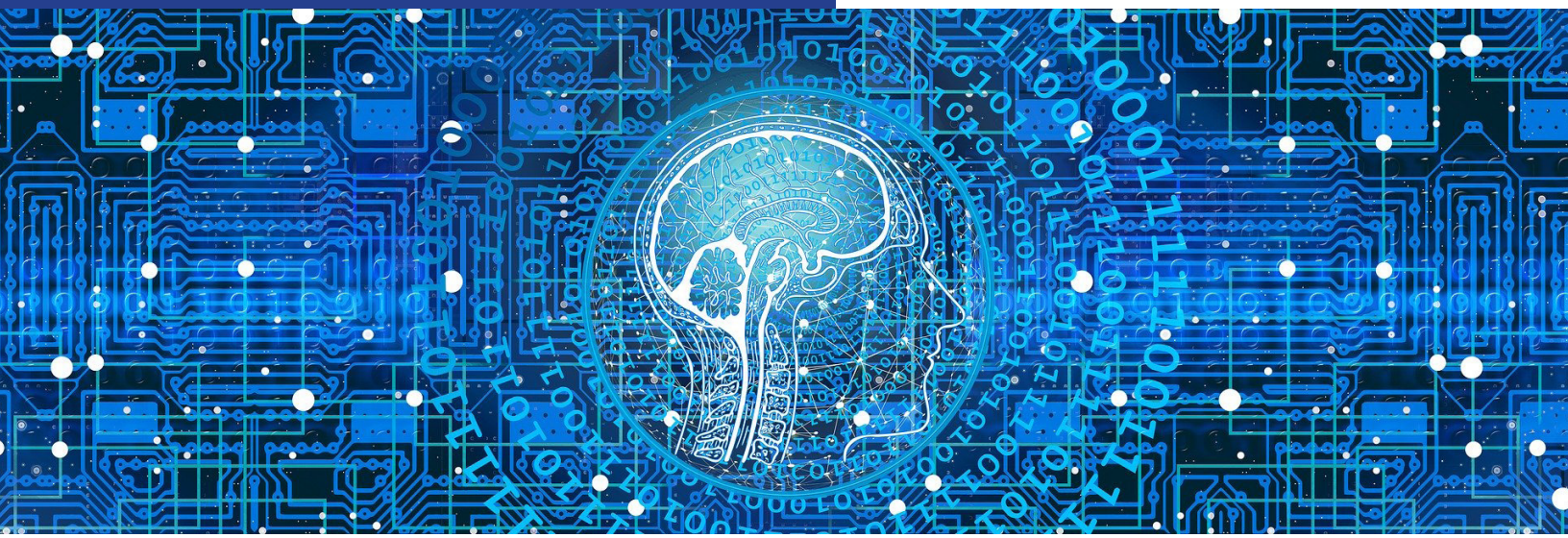
Edge Intelligence Workshop

HEC MONTRÉAL

March 2-3, 2020
Conference Proceedings



GERAD
GROUP FOR RESEARCH IN DECISION ANALYSIS

HUAWEI

NOAH'S ARK LAB

CANADA EXCELLENCE RESEARCH CHAIR
DATA SCIENCE FOR REAL-TIME DECISION-MAKING

CENTRE DE RECHERCHES MATHÉMATIQUES

Les Cahiers du GERAD ISSN: 0711–2440

Edge Intelligence Workshop 2020
Montreal, Canada, March 2–3, 2020
Papers

C. Audet, S. Le Digabel, A. Lodi,
V. Partovi Nia, (Eds.)

G–2020–23

April 2020

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2020
– Bibliothèque et Archives Canada, 2020

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2020
– Library and Archives Canada, 2020

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Editorial

Artificial Intelligence (AI) is the next society transformation builder. Massive AI-based applications include cloud servers, cell phones, cars, and pandemic management systems, among others. AI along with machine learning (ML) and deep learning (DL), beats human performance in various tasks, ranging from image classification, facial recognition, medical imaging, machine translation, speech recognition, and many more. Existing DL-based applications are computationally intensive, and require large amounts of resources, CPU, GPU, memory, and network bandwidth. These constraints restrict AI application deployment in practice. Embedded devices start with limited embedded memory in order of kilo bytes with limited computing power in order of 50 MHz, power in order of mili watts. Many of such edge devices still cannot support deployment of large large deep learning models.

Most voice assistants, such as Apple Siri, Google Voice, Microsoft's Cortana, or Huawei Celia, are based on cloud computing. Such services do not function if their network connection is slow or interrupted. Because existing intelligent applications rely on centralized data, users send their data to the cloud center and the computation is fully processed in the cloud. There is a big volume of data collected by billions of mobile end users and Internet of Thing (IoT) devices distributed at the network edge. Such giant data will explode by moving towards smart cities and 5G connectivity. According to recent forecasts, data generation by edge devices will reach 850 ZB by 2021. Providing such volume of data to the cloud requires large bandwidth resources and may violate users' privacy and the General Data Protection Regulation (GDPR) imposed by the European Union and adopted in many other countries as a data privacy standard.

Edge devices are the key enabler for modern AI applications. In recent years, we see a trend in homogenizing edge and IoT devices, in which billions of mobile and IoT devices are connected to the Internet, and generating a huge amount of data. AI for edge and IoT devices recently received significant attention leading to different synonyms, such as Edge AI, Edge Intelligence, Low Resource Computing, Energy Efficient Deep Learning, Embedded AI, among others. Industry and users both have considerable interest in keeping computation on edge. Industry saves computing resources by outsourcing computation to the user, and users gain privacy. However this privacy preservation is not free and AI consumers pay for their edge hardware. By pushing data storage, computing, analysis and control closer to the network edge, edge computing. Edge intelligence is also viewed as the ultimate solution to meet the requirements of latency, memory, scalability, energy efficiency, while resolving saving network bandwidth. In some applications edge computing is simply inevitable for robustness and safety reasons. For instance, in autonomous driving, constant high-quality network connectivity is an assumption rather than a guarantee.

Diverse edge applications such as smart cities, automation and massive device connectivity, pushes edge computing to deal with heterogeneous environments with edge big data. A wide range of AI fields will require AI-power at the edge. Edge intelligence is potentially the right tool to resolve many of the challenges that AI is faced today. This new interdisciplinary field, edge intelligence, requires to re-invent many of the AI models. Convolutional networks are mostly on point-wise depth-wise convolution in MobileNet type architectures sometimes combined with ResNet type connections to bring a high accuracy, while simplifying the computations to a great extent. AutoML focuses on searching architectures with reinforcement learning and evolutionary methods to design neural networks automatically that meet the computational constraint, NASNet, MNASNet, EfficientNet are quick examples. Most of these applications are focused on vision, though pushing the speech and language visions on edge is even more challenging because the neural architectures for such tasks are massively bigger and more complex.

Research on edge intelligence is still in its early stages, and a dedicated venue for exchanging the recent advances of edge intelligence is highly appealing especially in Canada which is one of the birthplaces of deep learning. There are many fundamental challenges faced to improve the neural architecture designed for certain edge devices that maximizes the use of heterogeneous computing architectures such as CPU, Embedded GPU, Neural Processing Units (NPU), available in a single IoT device. This workshop, the first of a series, calls for industry and academia experts to get together in an interdisciplinary environment that includes researchers from

- Mathematics, including functional analysis, numerical analysis, differential equations, linear algebra, operators theory, compressed sensing, topology;
- Optimization, including theoretical, numerical, blackbox, derivative-free, combinatorial, convex, linear, quadratic, and pure mathematics;
- Computer Science, including algorithm, machine learning, deep learning, artificial intelligence, computer vision, natural language processing, speech processing; Human Science including law, social science, ergonomics, economics;
- Medicine, including biomedical vision, drug discovery, biostatistics, epidemiology, etc.

Especially after the recent COVID–19 outbreak, sharing gathering and location information provides virus spread prediction on a large scale. This clearly shows the emergency of quick and reliable AI algorithms to control an epidemic.

Charles Audet, Sébastien Le Digabel, Andrea Lodi, Dominique Orban, and Vahid Partovi Nia
Editors

A word from the GERAD's director

GERAD is a proud co-organizer of the first Edge Intelligence Workshop held at HEC Montreal this past March 2020. This first event was a massive success; as seen not only by the profound quality of the research presented, but also by the great turnout of participants from both academia and the industry. The proceedings you are about to read are a vibrant testimony to this!

I hope they will inspire you to join the next Edge Intelligence Workshop. Enjoy your reading,

Olivier Bahn
Director, GERAD

Sponsoring societies

We are very grateful to our sponsors, whose financial contributions cover the expenses of the coffee breaks, lunches and a reception Without their help, the registration fees would have been significantly higher. These sponsors are the Centre de Recherches Mathématiques (CRM), the Canada Excellence Research Chair (CERC) on Data Science for Real-Time Decision Making the Groupe d'Études et de Recherche en Analyse de Décisions (GERAD), and Huawei.

Conference chairs

Andrea Lodi	CERC Science for Real-Time Decision Making & GERAD & Polytechnique Montréal
Vahid Partovi Nia	GERAD & Huawei Noah's Ark Lab

Editors

Charles Audet	GERAD & Polytechnique Montréal
Sébastien Le Digabel	GERAD & Polytechnique Montréal
Andrea Lodi	CERC Science for Real-Time Decision Making & GERAD & Polytechnique Montréal
Dominique Orban	GERAD & Polytechnique Montréal
Vahid Partovi Nia	GERAD & Huawei Noah's Ark Lab

Committees and organizers

Masoud Asgharian	McGill University
Charles Audet	GERAD & Polytechnique Montréal
Olivier Bahn	GERAD & HEC Montréal
Mark Coates	McGill University
Tiago Falk	INRS Montreal
Ali Ghodsi	University of Waterloo
Ashish Khisti	University of Toronto
Sébastien Le Digabel	GERAD & Polytechnique Montréal
Qun Liu	Huawei Noah's Ark Lab
Andrea Lodi	Science for Real-Time Decision Making & GERAD & Polytechnique Montréal
Alejandro Murua	Université de Montréal
Dominique Orban	GERAD & Polytechnique Montréal
Vahid Partovi Nia	GERAD & Huawei Noah's Ark Lab
Gennady Pekhimenko	University of Toronto
Mehdi Rezagholizadeh	Huawei Noah's Ark Lab
Fatiha Sadat	UQÀM
Shahrokh Valaee	University of Toronto
Chao Xing	Huawei Noah's Ark Lab

Local organization

Karine Hébert	GERAD (programm and proceedings)
Marilyne Lavoie	GERAD (graphism and website)
Marie Perreault	GERAD (registration, facilities, logistics and social events)
Nathalie Renault	GERAD (finance)

Contents

E. Sari, V. Partovi Nia
1 Batch normalization in quantized networks 8

G. Boukli Hacene, V. Gripon, M. Arzel, N. Farrugia, Y. Bengio
2 Pruning for efficient hardware implementations of deep neural networks 12

K.-A. Alahassa Nonvikan, A. Murua
3 Shallow Structured Potts Neural Network Regression (S-SPNNR) 17

Q. Tian, T. Arbel, J.J. Clark
4 Deep LDA-pruned nets and their robustness 23

S. Vicente, A. Murua
5 Statistical learning with the determinantal point process 29

G. Zhang, Y. Yu
6 Convergence of gradient methods on bilinear zero-sum games 39

X. Li, V. Partovi Nia
7 Random bias initialization improves quantized training 46

M. Zolnouri, X. Li, V. Partovi Nia
8 Importance of data loading pipeline in training deep neural networks 53

D. Robatian, M. Asgharian
9 Semi^{+}-supervised learning under sample selection bias 61

I. Amara, J. J. Clark
10 Uncertainty transfer with knowledge distillation 66

M. Shafiee, A. Hryniowski, F. Li, Z. Q. Lin, A. Wong
11 State of compact architecture search for deep neural networks 73

A. Abusitta, O. Abdel Wahab, T. Halabi
12 Deep learning for proactive cooperative malware detection system 79

A. Labach, S. Valaee
13 Neural network sparsification using Gibbs measures. 85

D. Mitra, A. Khisti
14 Distributed stochastic gradient descent with quantized compressive sensing 90

1 Batch normalization in quantized networks

Eyyüb Sari [a]

Vahid Partovi Nia [a,b]

[a] *Huawei Noah's Ark Lab, Montréal (Québec), Canada, H3N 1X9*

[b] *GERAD, HEC Montréal, Montréal (Québec), Canada, H3T 2A7*

eyyub.sari@huawei.com
vahid.partovinia@huawei.com

April 2020
Les Cahiers du GERAD
G-2020-23-EIW01

Abstract: *Implementation of quantized neural networks on computing hardware leads to considerable speed up and memory saving. However, quantized deep networks are difficult to train and batch normalization (BatchNorm) layer plays an important role in training full-precision and quantized networks. Most studies on BatchNorm are focused on full-precision networks, and there is little research in understanding BatchNorm affect in quantized training which we address here. We show BatchNorm avoids gradient explosion which is counter-intuitive and recently observed in numerical experiments by other researchers.*

1 Introduction

Deep Neural Networks (DNNs) compression through quantization is a recent direction in edge implementation of deep networks. Quantized networks are simple to deploy on hardware devices with constrained resources such as cell phones and IoT equipment. Quantized networks not only consume less memory and simplify computation, it also yields energy saving. Two well-known extreme quantization schemes are binary (one bit) and and ternary (two bit) networks, which allow up to $32\times$ and $16\times$ computation speed up, respectively. Binary quantization only keep track of the sign $\{-1, +1\}$ and ignores the magnitude, and ternary quantization extends the binary case to $\{-1, 0, +1\}$ to allow for sparse representation. BatchNorm facilitates neural networks training as a known fact. A common intuition suggests BatchNorm matches input and output first and second moments. There are two other clues among others: [4] claim that BatchNorm corrects covariate shift, and [6] show BatchNorm bounds the gradient and makes the optimization smoother in full-precision networks. None of these arguments work for quantized networks! The role of BatchNorm is to prevent exploding gradient empirically observed in [1] and [3].

2 Full-precision Network

Suppose a mini batch of size B for a given neuron k. Let $\hat{\mu}_k, \hat{\sigma}_k$ be the mean and the standard deviation of the dot product, between inputs and weights, $s_{bk}, b = 1, \dots B$. For a given layer l, BatchNorm is defined as $\mathrm{BN}(s_{bk}) \equiv z_{bk} = \gamma_k \hat{s}_{bk} + \beta_k$, where $\hat{s}_{bk} = \frac{s_{bk} - \hat{\mu}_k}{\hat{\sigma}_k}$ is the standardized dot product and the pair (γ_k, β_k) is trainable, initialized with $(1, 0)$.

Given the objective function $\mathcal{L}(.)$, BatchNorm parameters are trained in backpropagation

$$\frac{\partial \mathcal{L}}{\partial \beta_k} = \sum_{b=1}^{B} \frac{\partial \mathcal{L}}{\partial z_{bk}}, \quad \frac{\partial \mathcal{L}}{\partial \gamma_k} = \sum_{b=1}^{B} \frac{\partial \mathcal{L}}{\partial z_{bk}} \hat{s}_{bk},$$

For a given layer l, it is easy to prove $\frac{\partial \mathcal{L}}{\partial s_{bk}}$ equals

$$\frac{\gamma_k}{\hat{\sigma}_k} \Big(- \frac{1}{B} \sum_{b'=1}^{B} \frac{\partial \mathcal{L}}{\partial z_{b'k}} - \frac{\hat{s}_{bk}}{B} \sum_{b'=1}^{B} \frac{\partial \mathcal{L}}{\partial z_{b'k}} \hat{s}_{b'k} + \frac{\partial \mathcal{L}}{\partial z_{bk}} \Big). \tag{1}$$

Assume weights and activations are independent, and identically distributed (iid) and centred about zero. Formally, denote the dot product vector $\mathbf{s}_b^l \in \mathbb{R}^{K_l}$ of sample b in layer l, with K_l neurons. Let f be the element-wise activation function, $\mathbf{x}_b$ be the input vector, $\mathbf{W}^l \in \mathbb{R}^{K_{l-1} \times K_l}$ with elements $\mathbf{W}^l = [w_{kk'}^l]$ be the weights matrix; one may use w^l to denote an identically distributed elements of layer l. It is easy to verify

$$\frac{\partial \mathcal{L}}{\partial s_{bk}^l} \quad = \quad f'(s_{bk}^l) \sum_{k'=1}^{K_{l+1}} w_{kk'}^{l+1} \frac{\partial \mathcal{L}}{\partial s_{bk'}^{l+1}},$$

$$\frac{\partial \mathcal{L}}{\partial w^l_{k'k}} = \sum_{b=1}^{B} s^{l-1}_{bk'} \frac{\partial \mathcal{L}}{\partial s^l_{bk}}.$$

Assume that the feature element x and the weight element w are centred and iid. Reserve k to index the current neuron and use k' for the previous or the next layer neuron and where $\mathbb{V}(w^{l'})$ is the variance of the weight in layer l' $\mathbb{V}(s^l_{bk}) = \mathbb{V}(x)\prod_{l'=1}^{l-1} K_{l'}\,\mathbb{V}(w^{l'})$,

$$\mathbb{V}(\frac{\partial \mathcal{L}}{\partial s^l_{bk}}) = \mathbb{V}(\frac{\partial \mathcal{L}}{\partial s^L}) \prod_{l'=l+1}^{L} K_{l'}\,\mathbb{V}(w^{l'}),$$

which explodes or vanishes depending on $\mathbb{V}(w^{l'})$. This is the main reason common full-precision initialization methods suggest $\mathbb{V}(w^l) = \frac{1}{K_l}$. For any full-precision network, BatchNorm affects back-propagation as

$$\begin{aligned}\mathbb{V}\left(\frac{\partial \mathcal{L}}{\partial s^l_{bk}}\right) &= \left(\frac{\gamma^l_k}{B\hat{\sigma}^l_k}\right)^2 \{B^2 + 2B - 1 + \mathbb{V}(\hat{s}^{l^2}_{bk})\} \\ &\quad K_{l+1}\,\mathbb{V}(w^{l+1})\,\mathbb{V}\left(\frac{\partial \mathcal{L}}{\partial s^{l+1}}\right). \end{aligned} \tag{2}$$

3 Binary network

Controlling the variance has no fundamental effect on forward propagation if s_{bk} is symmetric about zero as the sign function filters the magnitude and only keeps the sign of the dot product. The term $b_k = \mu_k - \frac{\hat{\sigma}_k}{\gamma_k}\beta_k$ can be regarded as as a new trainable parameter, thus BatchNorm layer can be replaced by adding biases to the network to compensate. [7] shows that the gradient variance for binary quantized networks without BatchNorm is

$$\mathbb{V}\left(\frac{\partial \mathcal{L}}{\partial s^l_{bk}}\right) = \mathbb{V}\left(\frac{\partial \mathcal{L}}{\partial s^L}\right) \prod_{l'=l+1}^{L} K_{l'},$$

and with BatchNorm is

$$\mathbb{V}\left(\frac{\partial \mathcal{L}}{\partial s^l_{bk}}\right) = \prod_{l'=l}^{L-1} \frac{K_{l'+1}}{K_{l'-1}}\,\mathbb{V}\left(\frac{\partial \mathcal{L}}{\partial s^L}\right) + o\left(\frac{1}{B^{1-\epsilon}}\right),$$

for an arbitrary $0 < \epsilon < 1$.

Gradients are stabilized only if $\left(\frac{\gamma^l_k}{B}\right)^2 \{B^2 + 2B - 1 + \mathbb{V}(\hat{s}^{l^2}_{bk})\} \approx 1$. Moving from full-precision weight w to binary weight $\tilde{w} = \text{sign}(w)$ changes the situation dramatically: i) BatchNorm corrects exploding gradients in BNNs as the layer width ratio $\frac{K_{l+1}}{K_{l-1}} \approx 1$ in common neural models. If this ratio diverges from unity binary training is problematic even with BatchNorm.

4 Ternary network

Ternary neural networks (TNNs) are studied in [8] and the BatchNorm effect is detailed there. Full-precision weights during training are ternarized during forward propagation. Given a threshold Δ ternary quantization function is

$$\text{tern}(x) = \begin{cases} -1 & \text{if } x < -\Delta \\ +1 & \text{if } x > \Delta \\ 0 & \text{if } -\Delta \leq x \leq \Delta \end{cases} \tag{3}$$

Let's suppose the threshold is given so that the learning is feasible, for instance Δ is tuned so that $< 50\%$ of ternary weights are set to zero

$$\mathbb{V}(\tilde{w}_t^l) = 2p_1 = 1 - \frac{\Delta}{\sqrt{\frac{6}{K_l}}}. \tag{4}$$

In the literature [5] suggests to set $\Delta_l = 0.7\mathbb{E}(|w^l|)$. Under simplified assumptions of iid weight and activation

$$\Delta_l = \frac{0.7}{2}\sqrt{\frac{6}{K_l}} \tag{5}$$

and (4) reduces to $\mathbb{V}(\tilde{w}_t^l) = 1 - \frac{0.7}{2} = 0.65$. In this setting, variance is bigger than $\frac{2}{K_l}$ which produces exploding gradients similar to the binary case. Suppose weights and activation are iid and weights are centred about zero, for a layer l,

$$\hat{\sigma}_k^2 = K_{l-1}\frac{1}{2}\,\mathbb{V}(\hat{s}_b^{l-1})\,\mathbb{V}(\tilde{w}_t^l) = K_{l-1}\frac{1}{2}\,\mathbb{V}(\tilde{w}_t^l). \tag{6}$$

Therefore (2) reduces to

$$\mathbb{V}\left(\frac{\partial \mathcal{L}}{\partial s_{bk}^l}\right) = \left\{1 + o\left(\frac{1}{B^{1-\epsilon}}\right)\right\} \tag{7}$$

$$\frac{K_{l+1}}{K_{l-1}}\,\mathbb{V}\left(\frac{\partial \mathcal{L}}{\partial s^{l+1}}\right), \tag{8}$$

see [8] for details. Similar to the binary case, in most deep architectures $K_{l+1} \approx K_{l-1}$ or equivalently $\frac{K_{l+1}}{K_{l-1}} \approx 1$, so the variance would not explode for networks with BatchNorm layer.

5 Conclusion

We derived the analytical expression for full-precision network under assumptions of [2] and extended it for binary and ternary case. Our study shows that the real effect of BatchNorm is played in scaling. The main role of BatchNorm in quantized training is to adjust gradient explosion.

References

[1] Arash Ardakani, Zhengyun Ji, Sean C. Smithson, Brett H. Meyer, and Warren J. Gross. Learning recurrent binary/ternary weights. In International Conference on Learning Representations, 2019.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. CoRR, abs/1502.01852, 2015.

[3] Lu Hou, Jinhua Zhu, James Kwok, Fei Gao, Tao Qin, and Tie-yan Liu. Normalization helps training of quantized lstm. In Advances in Neural Information Processing Systems, pages 7344–7354, 2019.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR, abs/1502.03167, 2015.

[5] Fengfu Li and Bin Liu. Ternary weight networks. arXiv, abs/1605.04711, 2016.

[6] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems 31, pages 2483–2493. Curran Associates, Inc., 2018.

[7] Eyyüb Sari, Mouloud Belbahri, and Vahid Partovi Nia. How does batch normalization help binary training? arXiv preprint arXiv:1909.09139v2, 2019.

[8] Eyyüb Sari and Vahid Partovi Nia. Understanding batchnorm in ternary training. Journal of Computational Vision and Imaging Systems, 5(1):2, Jan. 2020.

2 Pruning for efficient hardware implementations of deep neural networks

Ghouthi Boukli Hacene
Vincent Gripon
Matthieu Arzel
Nicolas Farrugia
Yoshua Bengio

MILA/IMT Atlantique, Montréal (Québec), Canada, H2S 3H1

bouklihg@mila.quebec

April 2020
Les Cahiers du GERAD
G-2020-23-EIW02

Abstract: *Convolutional Neural Networks (CNNs) are state-of-the-art in numerous computer vision tasks such as object classification and detection. However, the large amount of parameters they contain leads to a high computational complexity and strongly limits their usability in budget-constrained mobile devices. In this paper, we propose a combination of a pruning technique and a quantization scheme that reduces complexity and memory of convolutional layers of CNNs, by replacing the complex convolutional operation by a low-cost multiplexer. We perform experiments on CIFAR10, CIFAR100, and SVHN and show that the proposed method achieves almost state-of-the-art accuracy, while drastically reducing the computational and memory footprint. We also propose an efficient hardware architecture to accelerate inference, which works as a pipeline and accommodates multiple layers working at the same time. In contrast with most proposed approaches that have used external memory or software defined memory controllers, our work is based on algorithmic optimization and full-hardware design.*

1 Introduction

For the past few years, Deep Neural Networks (DNNs), and especially Convolutional Neural Networks (CNNs) [7], have received considerable attention thanks to their remarkable accuracy in computer vision tasks [6, 8, 2, 9]. However, the need for intensive computations and memory leaded to the fact most DNN implementations are based on GPUs. Consequently, providing efficient hardware implementations is a very active and prospective field of research. Therefore, the deployment of CNNs in embedded systems is complex and potentially blocking for many applications.

In this paper, we propose to combine Shift Attention Layer (SAL) [3] a substitution to convolutional layers, in which the complex convolution operation is replaced by a shift operation followed by a multiplication with binary quantization of weights using Binary Weight Network (BWN), resulting in very lightweight DNNs in which complex convolution operations are replaced by low cost multiplexers that considerably eases hardware implementation on FPGA. We show in the following that such a combination approaches state-of-art accuracy while reducing computational and memory footprint. We also propose a hardware architecture in which we consider all processing blocks, memory blocks and controllers required, and implement such an architecture which uses very few resources and computational power on an FPGA, without using any external resources. This implementation can compute more than one layer at a time and uses a simple multiplexer to replace convolutional operations and process data through DNN layers. As such, it provides significantly smaller latency than existing counterparts, as shown in Section 3.

2 Shift layers and Shift Attention Layers

Let us denote by $\mathbf{x}$ (resp. $\mathbf{y}$ or $\mathbf{w}$) the input (resp. output or kernel) tensor of a given convolutional layer. We index $\mathbf{x}$ (resp. $\mathbf{y}$) using three indices i, j, k (resp. ℓ), where $0 \leq i < i_{\max}$ and $0 \leq j < j_{\max}$ correspond to 2D coordinates and $0 \leq k < k_{\max}$ (rsp. $0 \leq \ell < \ell_{\max}$) indexes a feature map. Similarly, we index $\mathbf{w}$ using four indices: $0 \leq \iota \leq \iota_{\max}$ and $0 \leq \lambda \leq \lambda_{\max}$ correspond to 2D coordinates, and k and ℓ are as introduced above. So, an element of the input tensor is written $x_{i,j,k}$, an element of the kernel tensor is written $w_{\iota,\lambda,k,l}$ and an element of the output tensor is written $y_{i,j,l}$.

To obtain a Shift Layer (SL) instead of a Convolutional Layer (CL), the authors in [11] propose to remove most of the connections in each slice $\mathbf{w}_{\cdot,\cdot,k,\ell}$ of the kernel tensor at the initialisation. The connections to be kept are chosen according to a deterministic rule agnostic of the initialization and of the training dataset. Namely, the authors choose to only keep the connections $w_{\iota,\lambda,k,\ell}$ for which

$$\iota + \lambda \iota_{\max} = k \pmod{\iota_{\max}\lambda_{\max}}. \tag{1}$$

When considering 3×3 kernels for example, 89% of the connections are pruned in the convolutional layer. Then, the training process is performed on remaining connections, disregarding the other ones.

Using such a configuration, all connections in a 3×3 kernel are pruned but one, and thus the convolution of each slice of the kernel tensor is replaced by a simple multiplication.

Contrary to SL where the kept connection is predetermined, SAL uses an attention mechanism [10] that selects the most relevant connection for each kernel and prunes the others. This process is performed during training, so that at the end of the training phase, the network configuration corresponds exactly to one obtained using SL.

To further benefit from the reduced complexity of the SAL method, we combine it with a weight binarization method such as binary weight network (BWN). Once remaining connections have been binarized, it is possible to replace the multiplication operation by a multiplexer, and thus, such a combination requires only low cost multiplexers to process data during inference.

Results

To evaluate the performance of our proposed combination, we use the CIFAR10 dataset. We compare various modern CNN architectures such as Resnet [4], Wide-Resnet [12] and Densenet [5]. Note that these architectures contain 1×1 and 3×3 convolutional kernels only. Thus we apply the proposed method on the 3×3 kernels.

We report in Table 1 the obtained results when using Equation (1) to remove kernels connections (SL), when applying SAL, and when combining SAL with BWN, and compare the accuracy obtained with baseline architectures. Note that BWN offers a 32 compression factor in terms of memory used, and SL or SAL method roughly multiply this factor by 9, achieving an almost 300 factor compression in total. Interstingly, we observe that when using SAL with BWN, the obtained accuracy remains at most 1% away to that of the baseline. We also perform experiments on SVHN (resp. CIFAR100) on Resnet18 and obtain 97%/96% (resp. 78%/75.2%) accuracy for Full-precision/SAL+BWN.

Table 1: Comparison of accuracy between baseline architectures, pruned ones, binarized ones, and the proposed method on CIFAR10.

	Resnet18	Resnet34	WideResnet-28-10	Densenet121
Full-precision	94.5%	95%	96.2%	95%
SL	93.5%	93.8%	95%	94.3%
SAL	94.2%	94.9%	96%	94.8%
SAL + BWN	93.5%	94.6%	95.4%	94.6%

3 Hardware implementation

The processing unit uses $\mathbf{X}$ (a feature vector, corrsponding to a row in a feature map) and a vector $\mathbf{W}$ made of P values coded on 1 bit each corresponding to weights. It thus computes in parallel P feature vectors (cf. Figure 1). The First-Input signal (FI) is set to 1 when the first feature vector is read from the BRAM to initialise registers by 0. To compute each feature vector p where $1 \leq p \leq P$, we use the corresponding W_p to add either $\mathbf{X}$ or $\mathbf{-X}$ to the content of register p. Once all input feature vectors have been read from the BRAM , the signal `Enable_s` is set to 1, and the content of registers is written one by one into the BRAM of the next layer. At the end of this process, the `Itter_done` signal is set to 1 in the processing unit block, so new data can be read to process other feature vectors.

To compute inference, $k_{max}j_{max}$ clock cycles (CCs) are required to copy all contents from a first layer's BRAM to a second layer's BRAM, $j_{max}k_{max}\ell_{max}/P$ CCs to compute all output feature vectors of one layer, and $j_{max}\ell_{max}$ CCs to write all computed feature vectors into the BRAM of a third layer. Thus the total number of CCs required is:

$$CCs = j_{max}k_{max} + \frac{j_{max}k_{max}\ell_{max}}{P} + j_{max}\ell_{max}. \tag{2}$$

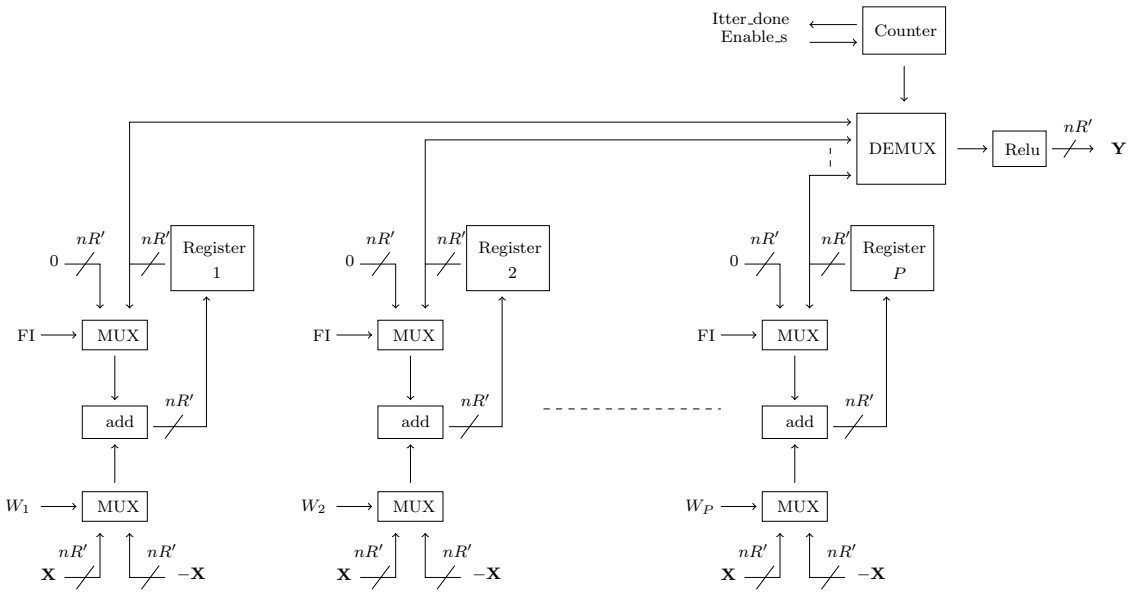


Figure 1: Hardware architecture of a processing unit block.

When comparing the total number of CCs of the proposed method with those obtained in [1], that are introduced by the following equation:

$$CCs = \frac{3j_{max}^2 k_{max} \ell_{max}}{P}, \tag{3}$$

we observe that the proposed architecture is $3j_{max}$ faster than [1], which can be significant when j_{max} is big. For instance with the CIFAR10 dataset, at the input layer of a CNN $j_{max} = 32$, and thus the proposed method is 96 times faster. In addition it is a pipeline architecture, so it can be $3Lj_{max}$ faster where L is the total number of layer blocks that fit in an FPGA.

Hardware results

We implemented one/few layers of Resnet18 on Xilinx Ultra Scale Vu13p (xcvu13p-figd2104-1-e) FPGA (c.f. Table 2). The implemented layers are arranged in a pipeline, and their functionality has been verified comparing the output of each layer block with the ones obtained by software simulation over a batch of examples.

Table 2: FPGA results for the proposed architecture on vu13p (xcvu13p-figd2104-1-e).

	P	LUT	FF	BRAMs	Frequency	Processing outflow	Power
Conv64 − 64	16	22424	22424	114	240MHz	19230 images/s	3.7W
4×Conv64 − 64	16	89746	75235	456	240MHz	19230 images/s	6.5W
3×Conv128 − 128	64	134090	102552	171	240MHz	29069 images/s	7.8W
3×Conv256 − 256	128	154599	102723	87	218MHz	26595 images/s	7.8W
3×Conv512 − 512	128	132155	52151	45	208MHz	16949 images/s	7.9W

4 Conclusion

In this paper, we proposed to reconsider the hardware implementation and acceleration of DNNs on limited resources embedded systems such as FPGA. We proposed to use lightweight DNNs architectures based on shift attention layers, and to combine them with weight binarization to reduce both complexity and memory usage. The resulting architecture almost matches the accuracy of considered baselines and only requires multiplexers, easing hardware implementation.

We implemented the proposed scheme using a low cost hardware architecture in which complex convolution operations are replaced by multiplexers. Thus, we were able to implement a considerable part of a complex CNNs (Resnet18). Moreover, the architecture only consumes a few watts and does not use any external ressources, making it a good solution for embedded applications.

References

[1] Arash Ardakani, Carlo Condo, and Warren J Gross. A convolutional accelerator for neural networks with binary weights. In Circuits and Systems (ISCAS), 2018 IEEE International Symposium on, pages 1–5. IEEE, 2018.

[2] Benjamin Graham. Fractional max-pooling. CoRR, abs/1412.6071, 2014.

[3] Ghouthi Boukli Hacene, Carlos Lassance, Vincent Gripon, Matthieu Courbariaux, and Yoshua Bengio. Attention based pruning for shift networks. arXiv preprint arXiv:1905.12300, 2019.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[5] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In CVPR, volume 1, page 3, 2017.

[6] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5 mb model size. arXiv preprint arXiv:1602.07360, 2016.

[7] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[8] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.

[9] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. arXiv preprint arXiv:1512.00567, 2015.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems, pages 5998–6008, 2017.

[11] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. arXiv preprint arXiv:1711.08141, 2017.

[12] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. arXiv preprint arXiv:1605.07146, 2016.

3 Shallow Structured Potts Neural Network Regression (S-SPNNR)

Karl-Augustt Alahassa Nonvikan
Alejandro Murua

Département de mathématiques et de statistique, Université de Montréal, Montréal (Québec), Canada, H3T 1J4

alahassan@dms.umontreal.ca

April 2020
Les Cahiers du GERAD
G-2020-23-EIW03

Abstract: *We introduce a novel ensemble learning approach which combines random partitions models through Potts clustering with a non-parametric predictor such as shallow feedforward neural networks (S-SPNNR). Neural network are known as universal approximators, and are very well suited to explore others learning methods. We combine them with Potts clustering models to create a bagging-like learning framework where several estimates from each random partition are aggregated into one prediction. Our approach carries out the balance between overfitting and model stability in presence of small datasets with high dimensional features. We found that S-SPNNR is really effective in* multivariate multiple regression *task and present more predictive power than Multi-layer feedforward neural network and the Multi-layer Multi-target Regression (MMR) model given some datasets from the Mulan Multi-label learning project.*

1 Introduction

The model called *Structured Potts Neural Network* is an hierarchical Bayesian model where we train individual neural nets to specialize on sub-groups (latent clusters components) while we still stay informed about representations of the overall data. Our Potts neural network model differ from those of [1] and [4], which is a generalization of the Ising neural network. We call it a structured one, because we integrate the structured correlations among the weights (and offsets) of the network [5] through Markov Random Fields (MRF) process. Bayesian learning allows the opportunity to quantify posterior uncertainty on neural networks (NNs) model parameters. We can specify priors to inform and constrain our models and get structured uncertainty estimation.

The proposal is organized as follows. Section 2 presents the background framework, section 3 explains and presents the model as well as its three variations: the Shallow- Structured Potts Neural Network Regression (S-SPNNR) with Sparse Markov Random Fields (ShallowSparse), the S-SPNNR with fully Connected Markov Random Fields (ShallowFull), and the S-SPNNR with compound symetry matrix (ShallowSym). Section 4 and 5 show our results and present our concluding remarks respectively.

2 Background

2.1 Potts clustering

We present Potts Clustering based on [3] paper framework. The training data consists of n examples in the form of inputs vector $x = x_i \in \mathbb{R}^q$, and corresponding outputs $y = y_i$, where $y_i \in \mathbb{R}^{l_2}$ (a vector response) for each $i = 1, ..., n$. For our model, $x = x_i$ is the vector of available covariates for observation i.

As in [3], we assume a random partition model with a hierarchical form for these data :

$$y_1, ..., y_n | \rho_n, \psi_1^*, ..., \psi_{k_n}^* \overset{\text{ind}}{\sim} p(y_i | x_i, \psi_{s_i}^*) \tag{1}$$

$$\psi_1^*, ..., \psi_{k_n}^* \overset{\text{ind}}{\sim} p(\psi) \tag{2}$$

$$\rho_n \sim p(\rho_n | x) \tag{3}$$

where ρ_n is a partition of $[n]$ into k_n subsets, $s_1, ..., s_n$ are cluster membership indicators such that $s_i = j$ if the ith individual belongs to the jth cluster, and $\psi_i = \psi_{s_i}^*$ represent the *neural network* parameters for all $i \in [n]$.

Potts clustering model can be seen as a stochastic version of the label propagation approach [6]. In following section, we present the *feed-forward* network function itself, which is of the form $y = g(x, w, b)$, with w weights matrix, b biases matrix (offsets), and g an activation function.

2.2 The feed-forward neural network regression framework

The network itself is (in general) a multi-layer network, defined typically by the following equations. Layer k computes an output vector h^k using the output h^{k-1} of the previous layer, starting with the input $x = h^0$.

$$h^k = b^k \oplus g_k(h^{k-1})w^k \tag{4}$$

with parameter b^k (a vector of offsets/biases), w^k a matrix of weights, $\oplus$ the Kronecker sum, and g_k which is applied element-wise, represents any suitable non-linear function.

The top layer output h^l is used for making a prediction and is combined with the supervised target y into a loss function $L(h^l, y)$. The model output y is given by :

$$\mathbb{E}[y|h^{l-1}] = b^l \oplus h^{l-1}w^l$$

In what follows, a 2-layer network means that we build two (2) layer on top of the input layer.

3 The models

3.1 The S-SPNNR model with Sparse Markov Random Fields (ShallowSparse)

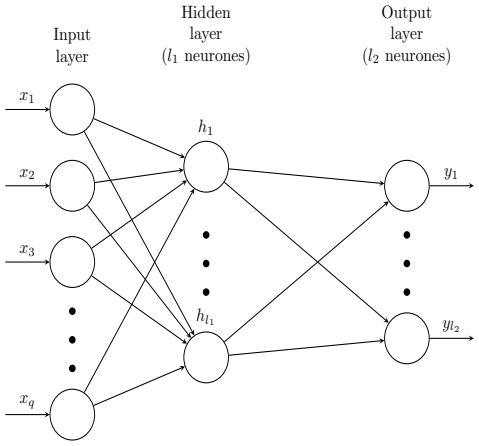


Figure 1: Shallow feedforward neural network

Given a Potts partition $\rho_n = (S_1, ..., S_{k_n})$ with k_n subsets, we denote by $\{\psi_1, \psi_2, ..., \psi_n\}$ the set of unique cluster-specific parameters. $y_j^* = \{y_i, i \in S_j\}$ and $x_j^* = \{x_i, i \in S_j\}$ denote respectively the set of responses and covariates of cluster S_j. Defining $h_i^2 = f_{\psi_j}(x_i)$, $h_{2j}^* = \{h_i^2, i \in S_j\}$.

$$p(y_j^*|h_{2j}^*, \psi_j, \Sigma) = \prod_{i \in S_j} (2\pi)^{-l_2/2}|\Sigma|^{-1/2} \times \exp\{-(1/2)(y_i - h_i^2)'\Sigma^{-1}(y_i - h_i^2)\} \tag{5}$$

with $\psi = (w^1, w^2, b^1, b^2)$ for each cluster. Our distribution specification for each y_i, $i = 1, ..., n$ is as follows:

$$\begin{aligned} y_i|x_i, \psi, \Sigma &\sim \mathcal{N}_{l_2}(f_\psi(x_i), \Sigma) \\ p(y_i|x_i, \psi, \Sigma) &= (2\pi)^{-l_2/2}|\Sigma|^{-1/2} \exp\{-(1/2)[y_i - f_\psi(x_i)]'\Sigma^{-1}[y_i - f_\psi(x_i)]\} \end{aligned} \tag{6}$$

The architecture in each cluster is a 2-layers network. The model weights uncertainty is similarly measured as in [5] paper. As [9] and [10] have introduced a deep-structured conditional random field model which consists of multiple layers of simple Conditional Random Fields (CRFs) where each layer's input consists of the previous layer's input and the resulting marginal probabilities. We use the Markov Random Fields (MRFs) to set alike structure on the neural network weights and biases.

The weights MG-MRF is sparse and defined on vector $w = (\text{vec}(w^1)^T, \text{vec}(w^2)^T)$, with mean $\mu_=(\mu_1^T, \mu_2^T)$ (let's say $\mu_k = \mathbb{E}[\text{vec}(w^k)]$), sparse precision matrix $\mathcal{J}$. For sparsity, we set only w_j^1 and w_i^2 as neighbors with $i = j$, where w_j^1 denotes the $j-$th column of w^1, and w_i^2 the $i-$th line of w^2.

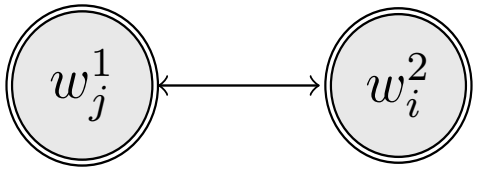


Figure 2: Sparse Multivariate Gaussian Markov Random Fields (MG-MRF) on the network weights

3.2 The S-SPNNR model with Fully Connected Markov Random Fields (ShallowFull)

The Fully Connected Markov Random Fields model is the same as described above with huge difference in weights connections. We set the whole matrices w^1 and w^2 as neighbors.

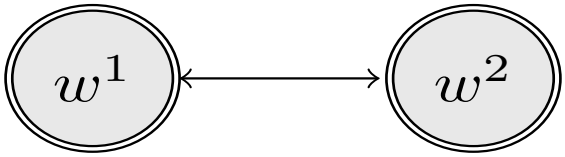


Figure 3: Fully Connected Markov Random Fields (MG-MRF) on the network weights

Fully-connected graphical models address issues of locally-connected models by assuming full connectivity amongst all nodes in the weights graph, thus taking full advantage of long range relationships to improve inference accuracy[8]. Just as importantly, in contrast to common fully-connected deep networks, we have less parameters in our case, thanks to the shallow network that present less connected layers.

3.3 The S-SPNNR-FCMRF model with compound symmetry matrix block (ShallowSym)

We have built for the Fully Connected Markov Random Field S-SPNNR model a compound symmetry version (ShallowSym) using the precision matrix $\mathcal{J}$. The matrix block $\mathcal{J}_{ii}$ for (w^1, w^2) itself can be express as a Kronecker product between two matrices U_i and V_i.

$$\mathcal{J}_{ii} = V_i \otimes U_i, U_i \in \mathcal{M}_{l_{i-1} \times l_{i-1}}, V_i \in \mathcal{M}_{l_i \times l_i}$$

To reduce the model complexity, we choose U_i and V_i to be a positive-definite matrix with compound symmetry structure (*constant diagonal and constant off-diagonal elements*). It means for example :

$$U_i = a_u I + (1 - \rho_u) \mathbb{1}\mathbb{1}^T$$

where a_u is a strictly positive number, and ρ_u a real-number. I is an identidy matrix with dimension l_{i-1}, and $\mathbb{1}$ a vector of ones of size l_{i-1}. In a more interpretive manner, a_u represent the intra-class correlation accross the weights and $a_u + (1 - \rho_u)$ their total variance [2] in the case V_i is estimated as a matrix of ones. This configuration is more likely usefull when all the variances may be nearly equal, and the covariances may be nearly equal among all the scalar weights at each layer. Those constraints save a lot of degrees of freedom with little loss of fit, because we only have to estimate one variance and one covariance for U_i.

4 Experimental evaluation

4.1 Datasets

The performance of the S-SPNNR in his three versions (ShallowSparse, ShallowFull and ShallowSym) were experimentally evaluated. The Mulan project [7] was used to evaluate the results. The experiments were performed on 11 multi-output regression datasets (see Table 1 below) that are among the benchmark data available from the Mulan project website.[1]

Table 1: Summary of data sets characteristics: name, domain, number of instances, features and targets

Data sets	Domain	Instances	Numb. of attributes	Numb. of targets
Andromeda	Water	49	30	6
Slump	Concrete	103	7	3
EDM	Machining	154	16	2
ATP7D	Forecast	296	211	6
ATP1D	Forecast	337	411	6
Jura	Geology	359	15	3
Online sales	Forecast	639	401	12
ENB	Buildings	768	8	2
Water quality	Biology	1 060	14	16
SCPF	Forecast	1 137	23	3
River flow 1	Forecast	9 125	64	8

We have also compared the performance of our models against the Multi-layer Multi-target Regression (MMR) model [11] that haved already substantially outperformed the best results from state-of-the-art algorithms on most of those 11 datasets and a 5-layer feedforward regression network (5-layer FFRNN).

To directly benchmark with state-of-the-art algorithms, we measure the performance by the commonly-used Relative Root Mean Squared Error (RRMSE) defined as :

$$\sqrt{\frac{\sum_{(xi,yi)\in Dtest}(\hat{y}_i - y_i)^2}{\sum_{(xi,yi)\in Dtest}(\hat{Y} - y_i)^2}}$$

where (x_i, y_i) is the i-th sample x_i with ground truth target y_i, $\hat{y}_i$ is the prediction of y_i and $\hat{Y}$ is the average of the targets over the training set $Dtrain$. We take the average RRMSE (aRRMSE) across all the target variables within the test set $Dtest$ as a single measurement. It measures the root squared error relative to what it would have been if a simple predictor had been used. A lower aRRMSE indicates better performance.

5 The results

Compare to a simple predictor, the proposed S-SPNNR model and its three versions have achieved great results against the MMR model. This large improvement of the proposed S-SPNNR over the MMR with significant margins on all the 11 datasets shows its effectiveness modeling multi-target regression task. Andromeda, and SCPF show that the 5-layer FFRNN is still beatable in terms of predictive power for these datasets. ShallowSparse was really effective on EDM, ATP7D, Jura, online sales and water quality against the ShallowFull and ShallowSym. ShallowSim was better against ShallowFull only on slump, ENB and water quality.

[1] http://mulan.sourceforge.net/datasets-mtr.html

Table 2: Summary of aRRMSE (%) obtained with S-SPNNR and MMR models

Data sets	‘ MMR	ShallowFull	ShallowSym
Andromeda	52.7	31.63	32.35
Slump	58.7	21.90	18.47
EDM	71.6	28.01	35.96
ATP7D*	44.3	22.69	24.56
ATP1D*	33.2	13.50	14.63
Jura	58.2	28.98	25.81
Online sales*	70.9	18.90	21.59
ENB*	11.1	39.05	45.79
Water quality	88.9	10.01	8.26
SCPF	81.2	12.30	13.86
River flow 1*	8.9	10.97	11.45

** We reduce the input features to the first 6 PCA components.*

Table 3: Summary of aRRMSE (%) obtained with S-SPNNR and the 5-layer FFRNN model

Data sets	‘ ShallowSparse	5-layer FFRNN
Andromeda	30.91	37.44
Slump	20.02	19.83
EDM	17.23	15.71
ATP7D*	19.73	13.67
ATP1D*	29.54	9.89
Jura	13.46	8.15
Online sales*	14.79	8.78
ENB*	23.92	4.36
Water quality	6.48	6.15
SCPF	10.78	18.49
River flow 1*	5.16	0.91

** We reduce the input features to the first 6 PCA components.*

References

[1] Ido Kanter. Potts-glass models of neural networks. Phys. Rev. A, 37:2739–2742, Apr 1988.

[2] Joris Mulder and Jean-Paul Fox. Bayesian tests on components of the compound symmetry covariance matrix. Statistics and Computing, 23, 01 2012.

[3] Alejandro Murua and Fernando A. Quintana. Semiparametric bayesian regression via potts model. Journal of Computational and Graphical Statistics, 26(2):265–274, 2017.

[4] WJM Philipsen and LJM Cluitmans. Using a genetic algorithm to tune potts neural networks. In Artificial Neural Nets and Genetic Algorithms, pages 650–657. Springer, 1993.

[5] Shengyang Sun, Changyou Chen, and Lawrence Carin. Learning Structured Weight Uncertainty in Bayesian Neural Networks. In Aarti Singh and Jerry Zhu, editors, Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, volume 54 of Proceedings of Machine Learning Research, pages 1283–1292, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR.

[6] Gergely Tibély and János Kertész. On the equivalence of the label propagation method of community detection and a potts model approach. Physica A: Statistical Mechanics and its Applications, 387(19):4982–4984, 2008.

[7] Grigorios Tsoumakas, Eleftherios Spyromitros-Xioufis, Jozef Vilcek, and Ioannis Vlahavas. Mulan: A java library for multi-label learning. Journal of Machine Learning Research, 12(Jul):2411–2414, 2011.

[8] Alexander Wong, Mohammad Javad Shafiee, Parthipan Siva, and Xiao Wang. A deep-structured fully connected random field model for structured inference. Access, IEEE, 3, 12 2014.

[9] Dong Yu, li Deng, and Shizhen Wang. Learning in the deep-structured conditional random fields. Proc. NeurIPS , vol. 1., pp. 1–8., 01 2009.

[10] Dong Yu, Shizhen Wang, and li Deng. Sequential labeling using deep-structured conditional random fields. Selected Topics in Signal Processing, IEEE Journal of, 4:965–973, 01 2011.

[11] Xiantong Zhen, Mengyang Yu, Xiaofei He, and Shuo Li. Multi-target regression via robust low-rank learning. IEEE transactions on pattern analysis and machine intelligence, 40(2):497–504, 2017.

4 Deep LDA-pruned nets and their robustness

Qing Tian
Tal Arbel
James J. Clark

McGill Centre for Intelligent Machines, McGill University, Montréal (Québec), Canada, H3A 2A7

qtian@cim.mcgill.ca
arbel@cim.mcgill.ca
clark@cim.mcgill.ca

April 2020
Les Cahiers du GERAD
G-2020-23-EIW04

Abstract: *Deep neural networks usually have unnecessarily high complexities and possibly many features of low utility, especially for tasks that they are not designed for. In this extended abstract, we present our Deep-LDA-based pruning framework as a solution to such problems. In addition to accuracy-complexity analysis, we investigate our approach's potential in improving networks' robustness against adversarial attacks (e.g. FGSM and NewtonFool Attacks) and noises (e.g. Gaussian, Poisson, Speckle). Experimental results on CIFAR100, Adience, and LFWA illustrate our framework's efficacy. Through pruning, we can derive smaller, but accurate and more robust models suitable for particular tasks.*

1 Introduction

With increasing network depths comes more complexity, which reignited research into network pruning. Approaches that sparsify networks by setting weights to zero include [7, 23, 21, 15, 6, 11, 24]. Compared to individual weights based approaches, filter or neuron pruning has its advantages. Instead of setting zeros in weights matrices, filter pruning removes rows/columns/depths in weight/convolution matrices, leading to direct space and computation savings [26, 17, 19, 9, 20, 27]. However, few if any works have investigated pruning's influence on model robustness. Given the large input-output dimension ratio, input-output correlation could be spurious. In this long abstract, we present our deep LDA based pruning and also analyze its influence on model robustness against adversarial attacks and noises. Through pruning large networks of high memorization capability, our method aims to help over-parameterized nets forget about task-unrelated factors and derive a feature subspace that is more invariant and robust to irrelevant factors and noises.

2 Deep Fisher LDA pruning

In this section, we present our deep Linear Discriminant Analysis (LDA) based pruning approach that pays direct attention to final task utility and its holistic cross-layer dependency. We define and capture task utility by deep LDA and use it to guide the pruning process. The approach is summarized as Algorithm 1.

Algorithm 1: Deep LDA Pruning of NN

Input: basenet, acceptable accuracy t_{acc}
Result: task-desirable pruned models
Pre-train: SGD optimization with cross entropy loss and dropout.
while *accuracy* $\geq t_{acc}$
do
  Step 1 → Pruning
    1. Penultimate LDA Utility Unravelling
    2. Cross-Layer Utility Tracing by Deconv
    3. Pruning as Utility Thresholding
  Step 2 → Re-training
    Similar to the pre-training step.
    Save model if needed.
end

In the rest of the section, we focus on the pruning step. Given the penultimate activation matrix X, our aim is to abandon dimensions of X that possess low or even negative task utility. Inspired by [4, 2, 29, 12, 1], Fisher's LDA is adopted to quantify this utility. Our goal of pruning is to find:

$$W_{opt} = \arg\max_{W} \frac{\mid W^T \Sigma_b W \mid}{\mid W^T \Sigma_w W \mid} \tag{1}$$

where Σ_w, Σ_b, Σ_a are within-class, between-class, and total scatter matrices. Through solving a generalized function with a decorrelated assumption of top layer motifs ([28]), we know that W columns are standard basis vectors. It follows that W columns and some of the original neuron dimensions are aligned. To maximize the class separation during pruning, we can safely discard neurons with small between-class to within-class variance ratios.

After unravelling twisted threads of deep variances and selecting dimensions of high LDA utility, the next step is to trace the utility across all previous layers to guide pruning. Inspired by [8], deconvolution as in [30] is used to reverse an unknown filter's effect and recover corrupted sources. One unit procedure is composed of unpooling, nonlinear rectification, and reversed convolution:

$$U_i = F_i^T Z_i \tag{2}$$

where i indicates the layer, Z_i is converted from feature maps, U_i is reconstructed contributing sources to final utility, F_i^T represents transposed convolution. Figure 1 provides a high level view of deep LDA cross-layer utility tracing. With all neurons'/filters' utility for final discriminability known, pruning simply becomes discarding structures that are less useful to final classification (colored white). Through pruning modular structures like Inception nets, the proposed approach determines how many filters, and of what types, are appropriate in a given layer. The threshold on utility is related to pruning rates. After pruning, retraining with surviving parameters is needed. Since our pruning selects filter dimensions according to task demands, the generated pruned models are more invariant to task-unrelated factors.

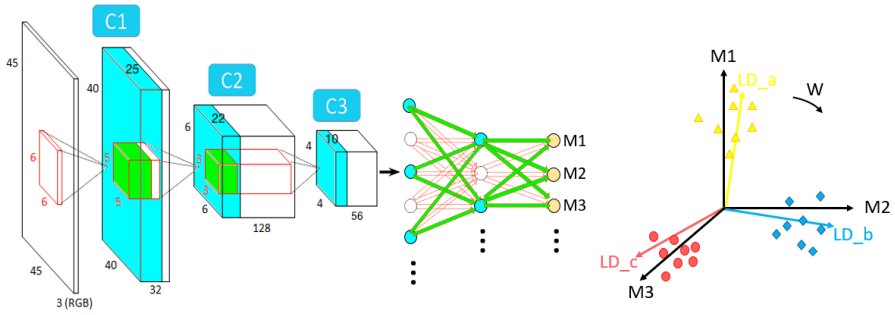
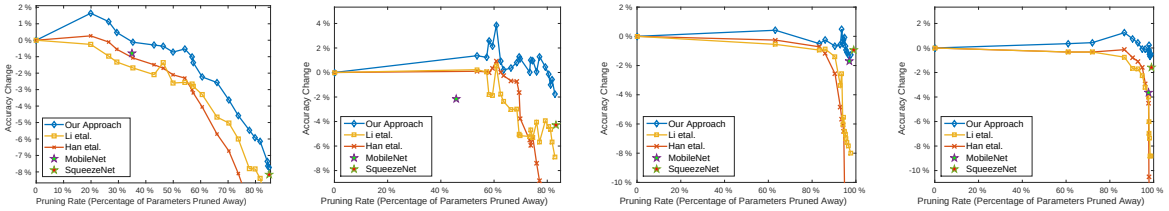


Figure 1: Deep LDA-Deconv utility tracing. Useful (cyan) neuron feature maps that contribute to final deep LDA utility through corresponding (green) next layer filter pieces, only depend on previous layers' (cyan) counterparts via deconv. our pruning leads to filter-wise and channel-wise savings simultaneously

3 Experiments and Results

We test our pruning approach on CIFAR100 [16], Adience [3], LFWA [18] datasets using conventional VGG16 [22] and modular Inception [25] as bases. All base networks are pretrained on ImageNet.



(a) CIFAR100, Inception **(b) Adience Age, Inception** **(c) LFWA Gender, VGG16** **(d) LFWA Smile, VGG16**

Figure 2: Accuracy change vs. parameters savings of our method (blue), [7] (red), and [17] (orange), SqueezeNet [13] and MobileNet [10] on validation data. Top-1 accuracy is used for CIFAR100

Figure 2 demonstrates the relationship of accuracy change v.s. parameters pruned. Pruning methods [7, 17] and compact structures [13, 10] are included for comparison. According to Figure 2,

even with large pruning rates (98-99% for the VGG16 cases, 57-82% for the Inception cases), our approach still maintains comparable accuracies to the original models (loss <1%), beating other pruning approaches. Compared to fixed nets, pruning offers the flexibility to find the boundary between over-fitting and over-compression.

In the rest of the section, we investigate our pruning's effects on the model's robustness to input perturbations. To this end, we apply Gaussian, Poisson, speckle noises and two adversarial attacks, i.e. FGSM [5] and Newton Fool Attack [14], to the testing data and compare how the original and pruned models perform in terms of accuracy drops (Table 1). The left subtable is for Inception and the right subtable is for VGG16 cases. The selected pruned model has similar accuracy to the unpruned one on the clean test set in each case. For fair comparison, the adversarial examples are generated on a third ResNet50 model. According to the results, the pruned models are more, or at least equally, robust to the noises than corresponding original unpruned models. One reason is that with fewer task-unrelated random filters, the pruned models are less likely to pick up irrelevant noises and are thus less vulnerable. Also, reducing parameters per se alleviates overfitting and thus brings down variance to data fluctuations. The deep nets are more prone to Gaussian and speckle noises than to Poisson noises. Furthermore, we can see that our pruning method also helps with model robustness to adversarial attacks. This is because fewer irrelevant deep feature dimensions can possibly mean fewer breaches where the adversarial attacks can easily put near-boundary samples to the other side of the decision boundary. That said, the pruning's effect on robustness is less obvious in the simple FGSM cases as compared to the Newton Fool Attack cases. Overall, both the task and the net architecture influence robustness. VGG16 and its pruned models are less susceptible to the attacks than Inception nets, perhaps because the adversarial examples are generated from ResNet50 in our case and are therefore more destructive to modular structures.

Table 1: Accuracy drops against noises and adversarial attacks for original and pruned nets (in the left table, the base is Inception net and in the right table, the base is VGG16). Note: Ori. means original nets, Gauss. represents Gaussian noise (stddev=5), speckle noise strength is 0.05. FGSM Attack: Fast Gradient Signed Method [5]. Newton Attack: Newton Fool Attack [14]

Acc Dif	CIFAR100		Adience		LFWA-G		LFWA-S	
	Ori.	Pruned	Ori.	Pruned	Ori.	Pruned	Ori.	Pruned
Gauss.	-2.5%	-2.0%	-0.5%	-0.1%	-5.2%	-4.2%	-1.4%	-1.2%
Poisson	-0.1%	0.0%	-0.3%	0.0%	0.0%	0.0%	0.0%	0.0%
Speckle	-3.7%	-3.1%	-1.5%	-1.0%	-0.5%	-0.2%	-0.2%	0.0%
FGSM	-8.1%	-7.4%	-0.4%	-0.4%	0.0%	0.0%	-0.1%	0.0%
Newton	-6.1%	-3.9%	-4.5%	-1.7%	-0.2%	-0.1%	-3.1%	-2.5%

Figure 3 and 4 illustrate some failure cases for the original unpruned nets and our pruned ones, respectively. Compared to the failed cases of pruned models in Figure 4, the fooled unpruned models in Figure 3 were usually very confident about their wrong predictions. The scenarios where our pruned models failed are usually ones where the pruned model was not very certain compared to the unpruned model even on the clean test data (e.g. girl vs woman, house vs castle, oak tree vs forest). Also, the nudges causing the pruned models to fail are usually more intuitive than those failed the unpruned models in Figure 3. For example, while it is not directly understandable how the attacks reverted the original model's predictions about smile/no smile (the two bottom left cases in Figure 3), we can see that the attack in the middle of the bottom row in Figure 4 attempted to lift up the mouth corner into a smile (best viewed when zoomed in). Both of the above observations are related to the fact that large network models remember more details than the pruned ones, thus can be more confident in prediction (either correct or wrong), but sensitive to intricate data fluctuation. On the other hand, to fool a compact model pruned according to task utility, the attack has to focus on remaining task-desirable dimensions since not many irrelevant, usually easily-fooled, loophole dimensions are present. In autonomous driving for example, to fool our pruned net to believe a red light to be green, the attacker possibly needs to literally change the color rather than apply some easy nuances.

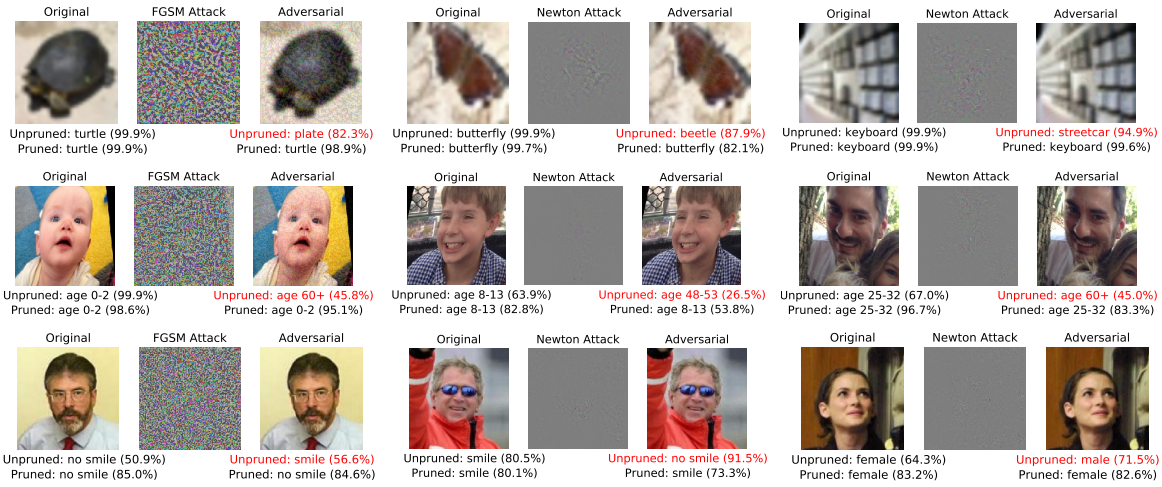


Figure 3: Example adversarial attacks that have fooled the original unpruned net, but not our pruned one

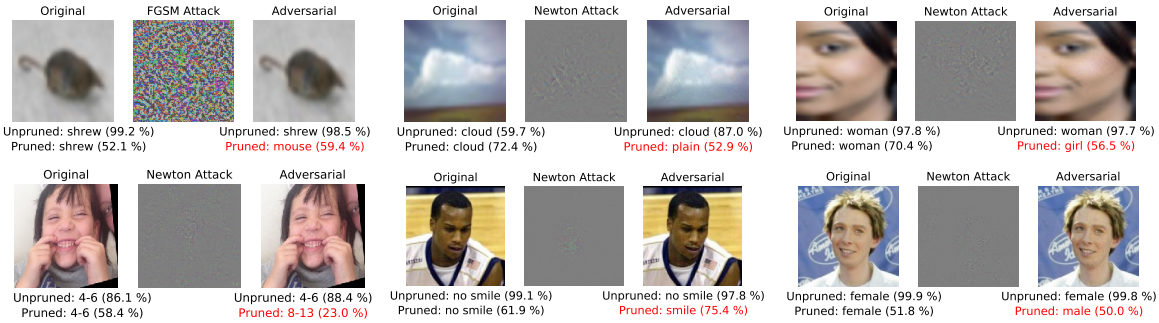


Figure 4: Example adversarial attacks that have fooled the pruned net, but not the original unpruned one

4 Conclusion

This paper presents deep-LDA based pruning that is aware of task utility and its cross-layer dependency. In addition to its high pruning rates, the method is shown to generate models that are more robust to adversarial attacks and noises than the unpruned one on the CIFAR100, LFWA and Adience datasets with VGG16 and Inception net as bases.

References

[1] Juan Bekios-Calfa, Jose M Buenaposada, and Luis Baumela. Revisiting linear discriminant techniques in gender recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 33(4):858–864, 2011.

[2] Peter N. Belhumeur, João P Hespanha, and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. IEEE Transactions on pattern analysis and machine intelligence, 19(7):711–720, 1997.

[3] Eran Eidinger, Roee Enbar, and Tal Hassner. Age and gender estimation of unfiltered faces. IEEE Transactions on Information Forensics and Security, 9(12):2170–2179, 2014.

[4] Ronald A Fisher. The use of multiple measurements in taxonomic problems. Annals of eugenics, 7(2):179–188, 1936.

[5] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572, 2014.

[6] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In Advances In Neural Information Processing Systems, pages 1379–1387, 2016.

[7] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In Advances in Neural Information Processing Systems, pages 1135–1143, 2015.

[8] Simon S Haykin. Blind deconvolution. Prentice Hall, 1994.

[9] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In Proceedings of the IEEE International Conference on Computer Vision, pages 1389–1397, 2017.

[10] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.

[11] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250, 2016.

[12] Gang Hua, Matthew Brown, and Simon Winder. Discriminant embedding for local image descriptors. In 2007 IEEE 11th International Conference on Computer Vision, pages 1–8. IEEE, 2007.

[13] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5 mb model size. arXiv preprint arXiv:1602.07360, 2016.

[14] Uyeong Jang, Xi Wu, and Somesh Jha. Objective metrics and gradient descent algorithms for adversarial examples in machine learning. In Proceedings of the 33rd Annual Computer Security Applications Conference, pages 262–277. ACM, 2017.

[15] Xiaojie Jin, Xiaotong Yuan, Jiashi Feng, and Shuicheng Yan. Training skinny deep neural networks with iterative hard thresholding methods. arXiv preprint arXiv:1607.05423, 2016.

[16] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[17] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710, 2016.

[18] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In Proceedings of International Conference on Computer Vision (ICCV), December 2015.

[19] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In Advances in Neural Information Processing Systems, pages 3288–3298, 2017.

[20] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In Proceedings of the IEEE international conference on computer vision, pages 5058–5066, 2017.

[21] Zelda Mariet and Suvrit Sra. Diversity networks. ICLR, 2016.

[22] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Proceedings of the International Conference on Learning Representations (ICLR) 2015, 2015.

[23] Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. arXiv preprint arXiv:1507.06149, 2015.

[24] Vivienne Sze, Tien-Ju Yang, and Yu-Hsin Chen. Designing energy-efficient convolutional neural networks using energy-aware pruning. pages 5687–5695, 2017.

[25] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1–9, 2015.

[26] Qing Tian, Tal Arbel, and James J Clark. Deep lda-pruned nets for efficient facial gender classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 10–19, 2017.

[27] Qing Tian, Tal Arbel, and James J Clark. Structured deep fisher pruning for efficient facial trait classification. Image and Vision Computing, 77:45–59, 2018.

[28] Qing Tian, Tal Arbel, and James J. Clark. Task-specific deep lda pruning of neural networks. arXiv preprint arXiv:1803.08134, 2018.

[29] Ming-Hsuan Yang. Kernel eigenfaces vs. kernel fisherfaces: Face recognition using kernel methods. In Fgr, volume 2, page 215, 2002.

[30] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In European Conference on Computer Vision, pages 818–833. Springer, 2014.

5 Statistical learning with the determinantal point process

Serge Vicente
Alejandro Murua

Département de mathématiques et de statistique, Université de Montréal, Montréal (Québec), Canada, H3T 1J4

vicentes@dms.umontreal.ca
murua@dms.umontreal.ca

April 2020
Les Cahiers du GERAD
G-2020-23-EIW05

Abstract: *The determinantal point process (DPP) provides a promising and attractive alternative to simple random sampling in cluster analysis or classification, for the initial random selection of points required by most algorithms. As a probabilistic model of repulsion, the DPP elects which points are similar and have less probability to appear together, favouring then more diverse subsets of points. After a short introduction to DPP, we show how its use for choosing initial subsets of points in a clustering algorithm run multiple times on large datasets can improve the quality of final results.*

1 Introduction

A classical core procedure in fields such as biology, psychology, medicine, marketing, computer vision and remote sensing is to group elements based on similar features (cluster analysis) [13], to provide a framework for learning. Some clustering techniques, such as the standard k-means algorithm or the partitioning around medoids (PAM) algorithm, are characterized by an initial choice of a subset of random points. We find the same type of initial choice in some classification techniques, such as neural networks or machine learning. However, selecting a simple random subset of points does not take into account the diversity among the selected points. As this type of sampling gives to every point an equal probability of being selected, a subset of points may include many similar points that carry the same type of information and representability. In some domains of research, where the diversity of elements is a major concern and ensures a better coverage of all its facets, single random sampling can lack some of them. The determinantal point process settles which points are similar and therefore have less probability to appear together, in contrast to simple random sampling. It intents to capture negative correlations between n points and has been used in machine learning as a model for subset selection [9]. Kulesza and Taskar [15] emphasize that the negative correlations are measured by a $n \times n$ matrix whose entries represent a measure of similarity between each pair of points. Similar elements have less probability to be co-selected, resulting in subsets that are more *diverse*. Clustering techniques in particular seek to obtain a unique optimal partition of data, by maximizing both intra-cluster similarity and inter-cluster dissimilarity. However, as stressed by [29], if different partitional techniques are applied to the same data, they can produce very different clustering results, due to the lack of an external objective and impartial criterion. The techniques' dependency on the initial choice of points can also explain those differences. To improve the quality and robustness of clustering results, [28] proposed the *cluster ensembles* framework, which main objective is to combine different clustering results into a single consolidated clustering. Monti et al. [21] introduced a cluster ensemble method in genomic studies and gene expression: *the consensus clustering*. Based on resampling and bootstrapping techniques, it seeks to attain a single consolidated clustering configuration from multiple runs of the same clustering algorithm. For sampling the initial points of the algorithm, we used the determinantal point process presented by [10, 15]. The paper is organized as follows: we present the determinantal point process in Section 2; we explain our consensus clustering algorithm in Section 3; we study the case of large datasets in Section 4; we present the quality measure for results in Section 5; we refer algorithms taken as reference in Section 6; we show results on simulated and real data in Section 7.

2 The determinantal point process (DPP)

Origins of DPP date back to [19] in quantum physics, known then as "fermion process", intended to model distributions of fermion systems at thermal equilibrium. The name "Determinantal Point Process" is established, introduced and made accepted as standard in mathematics' community by [3]. It also arises in studies of nonintersecting random paths, random spanning trees, and eigenvalues of random matrices [8, 4, 10].

Starting with a global overview of DPP, let $\mathcal{S} = \{\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}\}$ be a discrete set of n elements, with $2 \leq n < \infty$ and where $\boldsymbol{x_i}$ represents a $p-$dimensional vector, i.e. $\boldsymbol{x_i} \in \mathbb{R}^p$, $i = 1, \ldots, n$. A point process on $\mathcal{S}$ is a probability measure on $2^{\mathcal{S}}$, the set of all subsets of $\mathcal{S}$. It is called a DPP if, for a particular random subset $Y \in 2^{\mathcal{S}}$, its probability mass function is given by

$$P(\boldsymbol{Y} = Y) = \frac{\det(L_Y)}{\det(L + I_n)}, \tag{1}$$

where $\boldsymbol{Y}$ is a random variable representing the subset selected from $2^{\mathcal{S}}$, L is a $n \times n$ real, symmetric and positive semidefinite matrix measuring similarity between pair-wised elements of $\mathcal{S}$, L_Y is the submatrix of L with rows and columns indexed by Y, i.e., $L_Y = [L_{ij}]_{i,j \in Y}$ and I_n is the $n \times n$ identity matrix.

Determinants have a well-known geometric interpretation. Let B be a $m \times n$ matrix such that $L = B^T B$. B can always be found for $m < n$ due to positive semidefiniteness of L. Denoting the columns of B by B_i, for $i = 1, \ldots, n$, we have

$$P(\boldsymbol{Y} = Y) \propto \det(L_Y) = \mathrm{Vol}^2\left(\{B_i\}_{i \in Y}\right), \tag{2}$$

where Vol^2 represents the squared volume of the parallelepiped spanned by the columns of B corresponding to elements in Y. The columns of B can be interpreted as feature vectors describing the elements of $\mathcal{S}$ and, therefore, L measures similarity using dot products between feature vectors. By Equation (2), we can see the probability assigned by a DPP to a subset Y is related to the volume spanned by its associated feature vectors: diverse sets have then a higher probability, because their feature vectors are more orthogonal and hence span larger volumes.

3 Consensus clustering algorithm

Consider again the set $\mathcal{S}$ of n elements, and a particular partitional clustering technique run M times over the set $\mathcal{S}$. The agreement among the several runs of the algorithm is based on the consensus matrix C, a $n \times n$ symmetric matrix where the entry C_{ij}, $i, j = 1, \ldots, n$ represents the proportion of runs in which two elements $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ of $\mathcal{S}$ belong to the same cluster, i.e.

$$C_{ij} = \frac{\sum_{m=1}^{M} c_{ij}^m}{M}, \tag{3}$$

where c_{ij}^m is an indicator of wether element $\boldsymbol{x_i}$ belongs to the same cluster as $\boldsymbol{x_j}$ in the m-th run. The consensus clustering method was meant to attain a single consolidated clustering from multiple runs of the same clustering algorithm. Any partitional clustering method can be chosen, then. However, rather than using a well-known clustering method like $k-$means or PAM, we constructed our clustering algorithm to obtain a consolidated consensus clustering configuration. At each run, we start the algorithm with a Voronoi diagram on the set $\mathcal{S}$, which partitions the space into several cells or regions, based on a subset of points that are called generator points. These points will be sampled among the elements of $\mathcal{S}$ using the DPP defined by (1) and the sampling algorithm developed by [10, 15]. For the construction of the similarity matrix L, we will use kernel-based methods, which have been widely used in recent research into pattern analysis, like classification, regression and clustering [11]. As kernels are often considered measures of similarity, a higher kernel value represents a higher correlation in a high-dimensional (possibly infinite) Hilbert space. A popular kernel choice is the Gaussian kernel, which we will use to obtain the entries of L:

$$L = \left[\exp\left(-\frac{\|\boldsymbol{x_i} - \boldsymbol{x_j}\|^2}{2\sigma^2}\right)\right]_{i,j=1}^{n}, \tag{4}$$

where the scale parameter σ represents the relative spread of the distances $\|\boldsymbol{x_i} - \boldsymbol{x_j}\|$, the Euclidean distance between $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$, a common choice for the Gaussian kernel.

Finally, after M runs, we will obtain M Voronoi diagrams, from which we can compute the consensus matrix C with entries defined by (3). Following [2], if $C_{ij} \geq \theta$, with $0 \leq \theta \leq 1$, points $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ are defined as "friends" and then included in the same final cluster. As θ is unknown, there are several choices of final clusters, depending on the value of θ. Motivated then by [23] and [22], we used the *least-squares clustering* (LSCLUST) procedure of [7] to choose the optimal final cluster among the several choices: supposing we have B clusters, for each cluster $\boldsymbol{c}$ in $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_B$, a $n \times n$ matrix $\delta(\boldsymbol{c})$ can be built. The (i, j) element of the matrix, $\delta_{i,j}(\boldsymbol{c})$, is an indicator of wether element i of $\mathcal{S}$ belongs to the same cluster than j. Element-wise averaging of these association matrices yields a pairwise probability matrix of clustering, denoted $\widehat{\boldsymbol{\pi}}$. The least-squares clustering $\boldsymbol{c}_{LS}$ is the observed clustering $\boldsymbol{c}$ that solves the following minimization problem:

$$\boldsymbol{c}_{LS} = \underset{\boldsymbol{c} \in \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_B\}}{\arg\min} \sum_{i=1}^{n} \sum_{j=1}^{n} \left[\delta_{i,j}(\boldsymbol{c}) - \widehat{\boldsymbol{\pi}}_{i,j}\right]^2 .$$

4 Case of large datasets

The eigendecomposition of the matrix L of DPP defined by (1) is a central step for obtaining the generator points through the sampling algorithm of [10, 15]. It is well known that the computational complexity of eigendecomposition of a $n \times n$ symmetric matrix is $O\left(n^3\right)$ and, as n grows larger, the computation of the characteristic polynomial itself becomes expensive due to the computational complexity of calculating determinants. Therefore, computing only the largest eigenvalues can substantially reduce the computational burden of obtaining all the eigenvalues. The literature points out many references of well-known algorithms that can extract the t largest (or smallest) eigenvalues, with their associated eigenvectors, of a $n \times n$ Hermitian matrix, where usually, we have $t \ll n$. One of the most classical and used algorithms is the Lanczos algorithm [17] and its variations, such as the implicitly restarted Lanczos method, proposed by [5], which we will use if the dimension of L is very large. The Lanczos algorithm and its implicitly restarted variation were specially developed for large sparse symmetric matrices. Consequently, when L is large, to implement the implicitly restarted Lanczos method, it is necessary to find a good approximation of the dense matrix L by a sparse matrix. Nevertheless, the large size of L can still be a computational burden for the implementation of the algorithm, which motivated us to consider a special approach for dealing with large matrices, inspired by dimension reduction techniques.

Let then L be a large kernel matrix, of size $n \times n$, defined by (4) and let $L_1, L_2, \ldots, L_R$ denote a set of R submatrices of size $r \times r$ each, taken randomly from L, where $r < n$ (ideally, $r \ll n$). We apply the following methodology to the set of submatrices:

1. select randomly an index i_1 from $\{1, 2, \ldots, R\}$ and consider the submatrix L_{i_1};
2. find a sparse approximation of the submatrix L_{i_1} considering the k-nearest neighbours of each point of the submatrix, according to the k-nearest neighbours graph introduced by [25];
3. generate a sample Y_{i_1} from L_{i_1} through DPP, using the usual sampling algorithm of [10, 15] and the Lanczos algorithm for extracting the t largest eigenvalues;
4. build a Voronoi diagram for the n points, using the generated sample Y_{i_1};
5. repeat steps 1 to 4 for indexes $i_2, i_3, \ldots, i_N$, using always $\{1, 2, \ldots, R\}$;
6. apply the consensus clustering summarized in Section 3 to the set of N partitions obtained.

The number R of submatrices to be sampled must be chosen so that we get benefits from using the submatrices to sample the generator sets through DPP rather using the whole kernel matrix L. We know that the computational complexity of eigendecomposition of the $n \times n$ kernel matrix L is $O\left(n^3\right)$, employing n^3 operations. But, if we have R submatrices of size $r \times r$ each, the eigendecompositions of these submatrices employ Rr^3 operations. To obtain benefits from the sampled submatrices, we must guarantee that

$$\begin{aligned} Rr^3 &< < n^3 \Leftrightarrow \\ \Leftrightarrow R &< \left(\frac{n}{r}\right)^3 \Leftrightarrow \\ \Leftrightarrow R &< \left(\frac{1}{\gamma}\right)^3, \end{aligned}$$

where $\gamma = \frac{r}{n}$ represents the proportions of points considered for the submatrices. As we want to take advantage of dimension reduction and speed, we decided to choose R so that $R \ll \left(\frac{1}{\gamma}\right)^3$, and then, we decided to fix $R = \left\lfloor \frac{\left(\frac{1}{\gamma}\right)^3}{2} \right\rfloor$, where $\lfloor x \rfloor$ represents the floor function.

5 Clustering quality measure

As mentioned by [24], it is a common practice in the clustering literature to measure the goodness-of-fit of the optimal final cluster. Among the many known measures of goodness-of-fit that can be found in the literature, we will use the Adjusted Rand Index (ARI), first introduced by [26] and later adjusted for randomness by [12]. The ARI is a measure of agreement between two clustering configurations. The original Rand Index counts the proportion of elements that are either in the same clusters in both clustering configurations or in different clusters in both configurations. The adjusted version of the Rand Index corrected the calculus of the proportion, so its expected value is zero when the clustering configurations are random. The larger the ARI, the more similar the two configurations are, with the maximum ARI score of 1.0 indicating a perfect match.

6 Reference algorithms for comparison

To validate the performance of the consensus clustering summarized in Section 3 using DPP for choosing an initial set of points, we decided to compare the final results to two traditional clustering algorithms: PAM and k-means algorithms.

The PAM algorithm is a classical partitioning technique of clustering proposed by [14], which chooses data points for centers by simple random sampling. As DPP selects also data points for centers but based on diversity, the goal of comparing it with PAM method is to evaluate how the quality results of clustering behave if we consider diversity as a sampling criterion. The k-means algorithm was proposed by Stuart Lloyd in 1957, and later published in [18]. It starts with an initial set of k means, representing k clusters, assigning then each observation to the cluster with the nearest mean and proceeding with updating steps until convergence to a final optimal cluster configuration. However, as argued by [6], the popular methods for choosing the initial set of k means, such as Forgy, Random Partition and Maximin methods, result often in a final optimal cluster configuration with a low clustering quality. We decided then to use the k-means++ algorithm of [1], a popular choice that avoids the poor quality results of the traditional methods for choosing the initial means. Once more, our goal is to evaluate how the quality results of clustering with DPP behave if we consider diversity as a sampling criterion, when compared to the k-means algorithm that uses k-means++ for choosing initial points.

7 Results

The consensus clustering algorithm presented in Section 3 was applied to the case of datasets with a very large number of observations.

7.1 Simulated data

Using the algorithm of [20], we simulated a dataset constituted by $n = 10000$ vectors of dimension $p = 15$ grouped in 10 predefined clusters. As the dimension of the corresponding matrix L is very large, we decided to use the methodology described in Section 4 for the eigendecomposition required for the DPP sampling, adopting $N = 1000$. Prioritizing also a minimal computational time, we chose $\gamma = 0.1$ (and consequently $R = 500$), $k = 325$ nearest neighbours (which results in sparse matrices with approximately 60% of zeros) and the $t = 25$ largest eigenvalues of the sparse matrices. The consensus clustering algorithm of Section 3 was then applied, performing $M = 1000$ runs, and the quality of the optimal final cluster was assessed by the ARI described in Section 5. To ascertain the ARI variability, we decided to repeat the whole procedure 5 times and obtain one Boxplot for the ARI values. For comparisons, we also included the Boxplot resulting from the application of the clustering algorithm with DPP to the dense matrix L, from which we extracted all the eigenvalues, and the Boxplots resulting from k-means and PAM algorithms. All the comparing methods were also repeated 5 times for the construction of the Boxplots. Figure 1 presents the comparison of the four Boxplots.

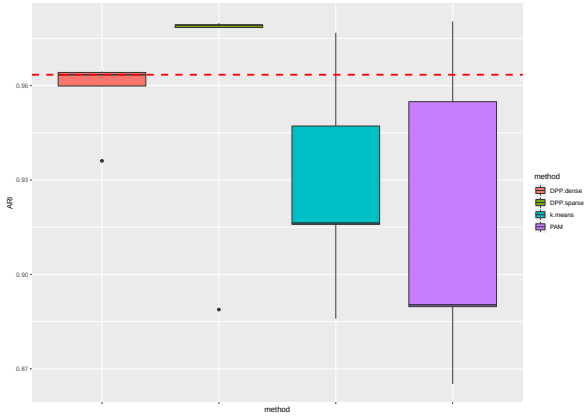


Figure 1: From left to right: Boxplots of the ARI for the DPP with dense matrix, DPP with sparse aproximations, k-means and PAM, with the dashed line indicating the median value obtained with the dense matrix

We also adopted another analysis to evaluate the quality of the sparse approximation of L: we obtain the kernel density estimation of the set of eigenvalues extracted from the sparse approximations of L and check graphically how the estimated density concentrates around all true eigenvalues of the dense kernel matrix L. We choose a Gaussian kernel for the density estimation and [27] rule for the bandwidth of the kernel. Figure 2 shows the estimated density and the values of the true eigenvalues.

We also took advantage of the opportunity to obtain the kernel density estimation of the set of the true eigenvalues extracted from the dense matrix L and measure its divergence from the kernel density estimation of the set of eigenvalues extracted from the sparse approximations of L depicted in Figure 2. The divergence will be measured through the Kullback-Leibler (KL) divergence, introduced by [16]. As the KL divergence does not obey to the symmetry property of a metric, for each pair of compared estimated densities, we will compute the KL divergence in both directions and compute the average of the two divergences. We can find the result in Table 1.

Table 1: KL divergence between the kernel density estimation of the eigenvalues extracted from the sparse approximations of L and the kernel density estimation of the true eigenvalues of L.

KL divergence
0.00005336

Additionally, we will also report and compare the elapsed time in seconds for eigenvalues computation using a sparse approximation of L or the original dense matrix L. The results are shown in Table 2.

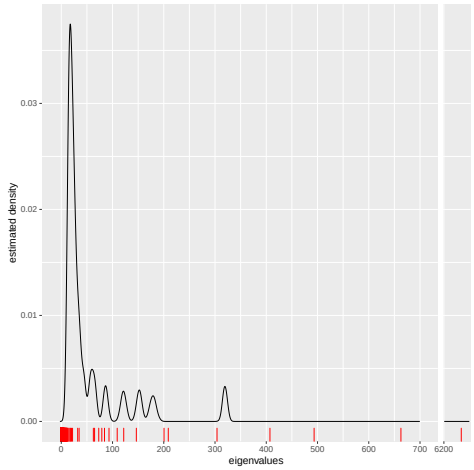


Figure 2: Kernel estimated density of the set of eigenvalues extracted from the sparse approximations of the dense matrix L, with the true eigenvalues marked on the abscissa axis

Table 2: Comparison of elapsed times (in seconds) for eigenvalues calculation.

	Elapsed time
Sparse L	0.079
Dense L	0.216

Finally, to explain the differences between DPP and PAM, we also present in Figure 3 the histograms of the logarithm of the probability mass function given by (1) for $M = 1000$ random subsets, using a DPP sampling or the simple random sampling.

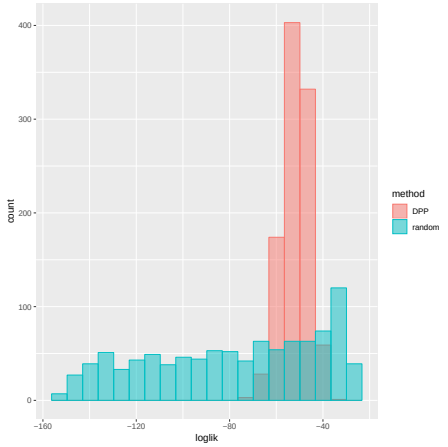


Figure 3: Histograms for the logarithm of the probability mass function (loglik) of $M = 1000$ random subsets using DPP and random sampling.

7.2 Real data

In this subsection, we considered two real datasets:

1. A dataset about human activity recognition and postural transitions using smartphones, collected from 30 subjects who performed six basic postures (downstairs, upstairs, walking, jogging, sitting and standing), including also six transitional postures between static postures (stand-to-sit, sit-to-stand, sit-to-lie, lie-to-sit, stand-to-lie and lie-to-stand), in the same environment and conditions, while carrying a waist-mounted smartphone with embedded inertial sensors. The

dataset consists of 10929 observations, with 561 time and frequency extracted features, which are commonly used in the field of human activity recognition. The dataset has then $n = 10929$ observations, $p = 561$ variables and $K = 12$ classes. The dataset is available on the *UCI Machine Learning Repository*, a well known database in the Machine Learning community for clustering and classification problems.

2. The Modified National Institute of Standards and Technology (MNIST) dataset, one of the most common datasets used for image classification. This dataset contains 60000 training images and 10000 testing images of handwritten digits, obtained from American Census Bureau employees and American high school students. Each observation represents a 28×28 pixel gray-scale image depicting a handwritten version of one of the ten possible digits (0 to 9). Pixels are organized row-wise, so that each row of the dataset represents an image, where the first number of each line is the label, i.e. the digit which is depicted in the image, and the remaining 784 numbers are the pixels of the 28×28 gray-scale image. The scale is available in two versions: the original scale between 0 (background or white) and 255 (foreground or black), or scaled between 0 and 1. For this section, we decided to use the testing set of 10000 images with the scaled pixels between 0 and 1. The dataset has then $n = 10000$ observations, $p = 784$ variables and $K = 10$ classes.

We applied the same strategy of Subsection 7.1 to each dataset, along with a comparison with k-means (with k-means++ for initial points) and PAM algorithms: we decided to sample a proportion $\gamma = 0.1$ of the points of the kernel matrix L (and consequently $R = 500$) and again obtain a sparse approximation of the sampled submatrices with 60% of sparsity, choosing the appropriate number k of nearest neighbours for each dataset. The Lanczos algorithm was then applied to extract the first $t = 25$ eigenvalues of the sparse approximated submatrices. The consensus clustering algorithm of Section 3 was then applied, performing $M = 1000$ runs, and the quality of the optimal final cluster was assessed by the ARI described in Section 5. To ascertain the ARI variability, we decided to repeat the whole procedure 5 times and obtain one Boxplot for the ARI values. For comparisons, we also included the Boxplots resulting from k-means and PAM algorithms. All the comparing methods were also repeated 5 times for the construction of the Boxplots. Figure 4 presents the results for the smartphones dataset and Figure 5 presents the results for the MNIST dataset.

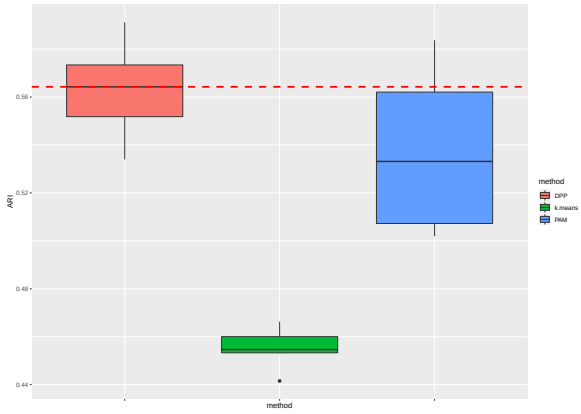


Figure 4: From left to right: Boxplots of the ARI for the DPP, k-means and PAM consensus clustering, with the dashed line indicating the median value obtained with DPP, for the smartphones dataset

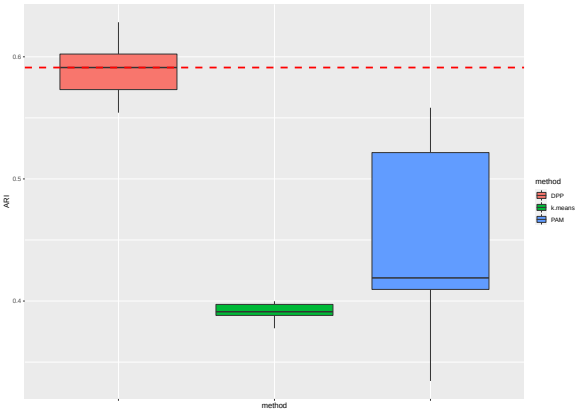


Figure 5: From left to right: Boxplots of the ARI for the DPP, k-means and PAM consensus clustering, with the dashed line indicating the median value obtained with DPP, for the MNIST dataset

8 Conclusion

The use of the sparse approximations is totally justified and has clear benefits, even with a low proportion γ of points sampled. The k-nearest neighbour graph approach provides then a good alternative to the use of the complete dense matrix L. Observing the results section, we present the following conclusions:

1. Simulated dataset: observing the Boxplots, we can see a lower quality of k-means and PAM results when compared to DPP. We also note that the approaches with DPP achieve a higher stability of the ARI, while the approaches with k-means and PAM give more heterogeneous results. Focusing on the kernel estimated density of the eigenvalues extracted from the sparse approximations of L, we can see a reasonably good concentration and fit around the true eigenvalues of L, supported by a low value of KL divergence. In terms of the elapsed time for eigenvalues extraction, we can see a clear time reduction, which becomes particularly important as the consensus clustering implies a repetition of the clustering algorithm a large number of times. Finally, the histograms of the logarithm of the probability mass function clearly show that DPP selects random subsets with higher and less dispersed probability mass values than simple random sampling, explaining a higher stability of the ARI.
2. Real datasets: observing the Boxplots, we can see a lower quality of k-means and PAM results when compared to DPP. We also note that the approaches with DPP achieve a higher stability of the ARI, while the approach with PAM give more heterogeneous results. Even if the k-means algorithm ensures a higher ARI stability when compared to DPP, it provides lower quality results.

The higher likelihood of the random subsets sampled by DPP confirms the higher diversity of those subsets, while the subsets sampled by random sampling can be highly or poorly diverse, with a very high dispersion in terms of diversity. DPP tends then to select points that maintain a high level of diversity at each sampling, proving then to be more consistent and stable than simple random sampling in terms of ensuring the heterogeneity of elements forming the subset. Moreover, taking into account the diversity of elements with DPP as a sampling method rather than simple random sampling, does not harm the quality of results, since the level attained by simple random sampling is more or less maintained, or even improved.

References

[1] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '07, page 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics.

[2] Marcelo Blatt, Shai Wiseman, and Eytan Domany. Superparamagnetic clustering of data. Phys. Rev. Lett., 76:3251–3254, Apr 1996.

[3] A. Borodin and G. Olshanski. Distributions on Partitions, Point Processes, and the Hypergeometric Kernel. Communications in Mathematical Physics, 211:335–358, 2000.

[4] Alexei Borodin and Alexander Soshnikov. Janossy densities i. determinantal ensembles. Journal of Statistical Physics, 113, 01 2003.

[5] Daniela Calvetti, L Reichel, and And D C Sorensen. An implicitly restarted lanczos method for large symmetric eigenvalue problems. Electronic Trans. Numer. Anal., 2:1–21, 04 1994.

[6] M. Emre Celebi, Hassan A. Kingravi, and Patricio A. Vela. A comparative study of efficient initialization methods for the k-means clustering algorithm. Expert Systems with Applications, 40(1):200–210, 2013.

[7] David B. Dahl. Model-Based Clustering for Expression Data via a Dirichlet Process Mixture Model, pages 201–218. Cambridge University Press, 2006.

[8] D. J. Daley and D. Vere-Jones. An introduction to the theory of point processes. Vol. I. Probability and its Applications (New York). Springer-Verlag, New York, second edition, 2003. Elementary theory and methods.

[9] R. Hafiz Affandi, E. B. Fox, and B. Taskar. Approximate Inference in Continuous Determinantal Point Processes. ArXiv e-prints, November 2013.

[10] J. Ben Hough, Manjunath Krishnapur, Yuval Peres, and Bálint Virág. Determinantal processes and independence. Probability Surveys, 3(0):206–229, 2006.

[11] Tom Howley and Michael G. Madden. An evolutionary approach to automatic kernel construction. In Stefanos Kollias, Andreas Stafylopatis, Włodzisław Duch, and Erkki Oja, editors, Artificial Neural Networks – ICANN 2006, pages 417–426, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[12] Lawrence Hubert and Phipps Arabie. Comparing partitions. Journal of Classification, 2(1):193–218, Dec 1985.

[13] Anil Jain and Richard Dubes. Algorithms for Clustering Data. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[14] Leonard Kaufmann and Peter Rousseeuw. Clustering by means of medoids. Data Analysis based on the L1-Norm and Related Methods, pages 405–416, 01 1987.

[15] A. Kulesza and B. Taskar. Determinantal point processes for machine learning. ArXiv e-prints, July 2012.

[16] S. Kullback and R. A. Leibler. On information and sufficiency. Ann. Math. Statist., 22(1):79–86, 03 1951.

[17] Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. Journal of Research of the National Bureau of Standards, 1950.

[18] Stuart P. Lloyd. Least squares quantization in pcm. IEEE Trans. Inf. Theory, 28:129–136, 1982.

[19] Odile Macchi. The Coincidence Approach to Stochastic Point Processes. Advances in Applied Probability, 7(1):83–122, 1975.

[20] Volodymyr Melnykov, Wei-Chen Chen, and Ranjan Maitra. Mixsim: An r package for simulating data to study performance of clustering algorithms. Journal of Statistical Software, Articles, 51(12):1–25, 2012.

[21] Stefano Monti, Pablo Tamayo, Jill Mesirov, and Todd Golub. Consensus clustering: A resampling-based method for class discovery and visualization of gene expression microarray data. Machine Learning, 52(1):91–118, 2003.

[22] Johanna Muñoz and Alejandro Murua. Building cancer prognosis systems with survival function clusters. Statistical Analysis and Data Mining: The ASA Data Science Journal, 11(3):98–110, 2018.

[23] Alejandro Murua and Fernando A. Quintana. Semiparametric bayesian regression via potts model. Journal of Computational and Graphical Statistics, 26(2):265–274, 2017.

[24] Alejandro Murua and Nicolas Wicker. The Conditional-Potts Clustering Model. Journal of Computational and Graphical Statistics, 23(3):717–739, 2014.

[25] Rodrigo Paredes and Edgar Chávez. Using the k-nearest neighbor graph for proximity searching in metric spaces. In Proceedings of the 12th International Conference on String Processing and Information Retrieval, SPIRE'05, pages 127–138, Berlin, Heidelberg, 2005. Springer-Verlag.

[26] William M. Rand. Objective criteria for the evaluation of clustering methods. Journal of the American Statistical Association, 66(336):846–850, 1971.

[27] B. W. Silverman. Density estimation for statistics and data analysis, volume 26 of Monographs on Statistics & Applied Probability. Chapman and Hall, 1986.

[28] Alexander Strehl and Joydeep Ghosh. Cluster ensembles — a knowledge reuse framework for combining multiple partitions. J. Mach. Learn. Res., 3:583–617, 2002.

[29] Sandro Vega-Pons and José Ruiz-Shulcloper. A survey of clustering ensemble algorithms. International Journal of Pattern Recognition and Artificial Intelligence, 25(03):337–372, 2011.

6 Convergence of gradient methods on bilinear zero-sum games

Guojun Zhang
Yaoliang Yu

University of Waterloo & Waterloo AI Institute, Waterloo (Ontario), Canada, N2L 3G1

Vector Institute MaRS Centre, Toronto (Ontario), Canada, M5G 1M1

guojun.zhang@uwaterloo.ca

April 2020
Les Cahiers du GERAD
G-2020-23-EIW06

Abstract: *Min-max formulations have attracted great attention in the ML community due to the rise of deep generative models and adversarial methods, while understanding the dynamics of gradient algorithms for solving such formulations has remained a grand challenge. As a first step, we restrict to bilinear zero-sum games and give a systematic analysis of popular gradient updates, for both simultaneous and alternating versions. We provide exact conditions for their convergence and find the optimal parameter setup and convergence rates. In particular, our results offer formal evidence that alternating updates converge "better" than simultaneous ones.*[1]

1 Introduction

Min-max optimization has received significant attention due to the popularity of generative adversarial networks (GANs) [14], adversarial training [19] and reinforcement learning [8], just to name some examples. Formally, given a (bivariate) objective function $f(\mathbf{x}, \mathbf{y})$, we aim to find a *saddle point* $(\mathbf{x}^*, \mathbf{y}^*)$ such that

$$f(\mathbf{x}^*, \mathbf{y}) \leq f(\mathbf{x}^*, \mathbf{y}^*) \leq f(\mathbf{x}, \mathbf{y}^*),$$

$\forall \mathbf{x} \in \mathbb{R}^{\mathbf{n}}, \forall \mathbf{y} \in \mathbb{R}^{\mathbf{n}}$. Since the beginning of game theory, various algorithms have been proposed for finding saddle points [2, 7, 13, 16, 26, 3, 18, 22, 9]. Due to its recent resurgence in ML, new algorithms designed for training GANs were proposed [5, 15, 11, 20]. However, due to non-convexity in deep learning formulations, our understanding of the convergence behaviour of new and classic gradient algorithms is still limited, and existing analysis mostly focused on bilinear games [5, 11] or strongly-convex-strongly-concave games [17, 21, 29]. Non-zero-sum bilinear games, on the other hand, are PPAD-complete [4] (for the definition see [24]; for finding approximate Nash equilibria, see e.g. [6]).

In this work, we focus on bilinear zero-sum games as a first step towards understanding general min-max optimization, although our results apply to some simple GAN settings [10]. It is well-known that certain gradient algorithms converge at a linear rate on bilinear zero-sum games [17, 21, 26, 16]. These iterative algorithms usually come with two versions: *Jacobi* style or *Gauss–Seidel* (GS) style. In Jacobi style, we update the two sets of parameters (i.e., $\mathbf{x}$ and $\mathbf{y}$) *simultaneously* whereas in GS style we update them *alternatingly* (i.e., one after the other). Thus, Jacobi style updates are naturally amenable to parallelization while GS style updates have to be sequential, although the latter are usually found to converge faster (and more stable). In numerical linear algebra, the celebrated Stein–Rosenberg theorem [28] formally proves that in solving certain linear systems, GS updates converge *strictly* faster than their Jacobi counterparts, and often with a larger set of convergent instances. However, this result does not readily apply to bilinear zero-sum games (see Section 3). Our main goal here is to answer the following questions about solving bilinear zero-sum games:

- When exactly does a gradient-type algorithm converge?
- What is the optimal convergence rate by tuning the step size or other parameters?
- Can we prove similar things for Jacobi and GS updates as the Stein–Rosenberg theorem?

Contributions In Section 2, we review bilinear games and popular gradient algorithms. On bilinear games, gradient algorithms have a unified formulation. With this new formulation, we give exact convergence conditions, and show that alternating updates are more stable than their simultaneous counterparts in Section 3. We give optimal convergence rates for different algorithms in Section 4 with supporting experiments in Section 5.

[1] A more thorough version is published at ICLR 2020 [30].

2 Preliminaries

Mathematically, zero-sum *bilinear* games can be formulated as the following min-max problem:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \max_{\mathbf{y}\in\mathbb{R}^n} \quad \mathbf{x}^\top \mathbf{E}\mathbf{y} + \mathbf{b}^\top \mathbf{x} + \mathbf{c}^\top \mathbf{y}. \tag{1}$$

(Throughout for simplicity we assume $\mathbf{E}$ is invertible.) For bilinear games, it is well-known that simultaneous gradient descent does not converge [22] and other gradient-based algorithms tailored for min-max optimization have been proposed [16, 5, 10, 20]. These iterative algorithms all belong to the class of general linear dynamical systems (LDSs), and they can be described as:

$$\mathbf{z}^{(t)} = \textstyle\sum_{i=1}^k \mathbf{A}_i \mathbf{z}^{(t-i)} + \mathbf{d}, \quad \mathbf{z}^{(t)} := (\mathbf{x}^{(t)}, \mathbf{y}^{(t)}).$$

The following well-known result decides when such a k-step LDS converges for any initialization:

Theorem 1 (e.g. [12]) *The LDS* $\mathbf{z}^{(t)} = \sum_{i=1}^k \mathbf{A}_i \mathbf{z}^{(t-i)} + \mathbf{d}$ *converges for any initialization* $(\mathbf{z}^{(0)}, \dots, \mathbf{z}^{(k-1)})$ *iff the spectral radius* $r := \max\{|\lambda| : \det(\lambda^k \mathbf{I} - \sum_{i=1}^k \mathbf{A}_i \lambda^{k-i}) = 0\} < 1$*, in which case* $\{\mathbf{z}^{(t)}\}$ *converges linearly with (asymptotic) exponent* r*.*

Therefore, understanding the bilinear game dynamics reduces to spectral analysis. The (sufficient and necessary) convergence condition reduces to that all roots of the characteristic polynomial lie in the unit circle, which can be conveniently analyzed through the celebrated Schur's theorem [27].

Let us formally define Jacobi and GS updates: Jacobi updates take the form

$$\mathbf{x}^{(t)} = T_1(\mathbf{x}^{(t-1)}, \mathbf{y}^{(t-1)}, \dots, \mathbf{x}^{(t-k)}, \mathbf{y}^{(t-k)}),$$
$$\mathbf{y}^{(t)} = T_2(\mathbf{x}^{(t-1)}, \mathbf{y}^{(t-1)}, \dots, \mathbf{x}^{(t-k)}, \mathbf{y}^{(t-k)}),$$

while Gauss–Seidel updates replace $\mathbf{x}^{(t-i)}$ with the more recent $\mathbf{x}^{(t-i+1)}$ in operator T_2, where $T_1, T_2 : \mathbb{R}^{nk} \times \mathbb{R}^{nk} \to \mathbb{R}^n$ can be any update functions. For LDS updates in (2) we find a nice relation between the characteristic polynomials of Jacobi and GS updates:

Theorem 2 (Jacobi vs. Gauss–Seidel) *Let* $p(\lambda, \gamma) = \det(\sum_{i=1}^k (\gamma \mathbf{L}_i + \mathbf{U}_i)\lambda^{k-i} - \lambda^k \mathbf{I})$*, where* $\mathbf{A}_i = \mathbf{L}_i + \mathbf{U}_i$ *and* $\mathbf{L}_i$ *is strictly lower block triangular. Then, the characteristic polynomial of the Jacobi update is* $p(\lambda, 1)$ *while that of the Gauss–Seidel update is* $p(\lambda, \lambda)$*.*

Next, we define some popular gradient algorithms for finding saddle points in the min-max problem $\min_x \max_y f(\mathbf{x}, \mathbf{y})$. Unlike their usual presentations, we introduced more "step sizes" for refined analysis, as the enlarged parameter space often contain choices for faster linear convergence (see Section 4). We only define the Jacobi updates, while the GS counterparts can be easily inferred.

Extra-gradient (EG) We study a generalized version of EG, defined as follows:

$$\mathbf{x}^{(t+1/2)} = \mathbf{x}^{(t)} - \gamma_2 \nabla_{\mathbf{x}} f(\mathbf{x}^{(t)}, \mathbf{y}^{(t)}), \qquad \mathbf{y}^{(t+1/2)} = \mathbf{y}^{(t)} + \gamma_1 \nabla_{\mathbf{y}} f(\mathbf{x}^{(t)}, \mathbf{y}^{(t)}); \tag{2}$$
$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha_1 \nabla_{\mathbf{x}} f(\mathbf{x}^{(t+1/2)}, \mathbf{y}^{(t+1/2)}), \qquad \mathbf{y}^{(t+1)} = \mathbf{y}^{(t)} + \alpha_2 \nabla_{\mathbf{y}} f(\mathbf{x}^{(t+1/2)}, \mathbf{y}^{(t+1/2)}). \tag{3}$$

EG was first proposed in [16] with the restriction $\alpha_1 = \alpha_2 = \gamma_1 = \gamma_2$, under which linear convergence was proved for bilinear games. A slightly more generalized version was analyzed in [17] where $\alpha_1 = \alpha_2$, $\gamma_1 = \gamma_2$, again with linear convergence proved. For later convenience we define $\beta_i = \alpha_i \gamma_i$.

Optimistic gradient descent (OGD) We study a generalized version of OGD, defined as follows:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha_1 \nabla_{\mathbf{x}} f(\mathbf{x}^{(t)}, \mathbf{y}^{(t)}) + \beta_1 \nabla_{\mathbf{x}} f(\mathbf{x}^{(t-1)}, \mathbf{y}^{(t-1)}), \tag{4}$$
$$\mathbf{y}^{(t+1)} = \mathbf{y}^{(t)} + \alpha_2 \nabla_{\mathbf{y}} f(\mathbf{x}^{(t)}, \mathbf{y}^{(t)}) - \beta_2 \nabla_{\mathbf{y}} f(\mathbf{x}^{(t-1)}, \mathbf{y}^{(t-1)}). \tag{5}$$

The original version of OGD was given in [5] with $\alpha_1 = \alpha_2 = 2\beta_1 = 2\beta_2$, and its linear convergence for bilinear games was proved in [17]. A slightly generalized version with $\alpha_1 = \alpha_2$ and $\beta_1 = \beta_2$ was analyzed in [21], again with linear convergence proved.

Momentum method Generalized heavy ball method was proposed and analyzed in [11]:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha_1 \nabla_{\mathbf{x}} f(\mathbf{x}^{(t)}, \mathbf{y}^{(t)}) + \beta_1(\mathbf{x}^{(t)} - \mathbf{x}^{(t-1)}), \tag{6}$$

$$\mathbf{y}^{(t+1)} = \mathbf{y}^{(t)} + \alpha_2 \nabla_{\mathbf{y}} f(\mathbf{x}^{(t)}, \mathbf{y}^{(t)}) + \beta_2(\mathbf{y}^{(t)} - \mathbf{y}^{(t-1)}), \tag{7}$$

as a modification of Polyak's heavy ball (HB) [25], which also motivated Nesterov's accelerated gradient algorithm (NAG) [23]. For bilinear games, HB and NAG are the same and hence we call both the momentum method. For this algorithm our result below improves those obtained in [11].

3 Exact conditions

With tools from Section 2, we give necessary and sufficient conditions under which a gradient-based algorithm converges for bilinear games. For simplicity, we mostly take the parameters for the two sets of variables to be the same, i.e., $\alpha_1 = \alpha_2 = \alpha$, $\beta_1 = \beta_2 = \beta$ and $\gamma_1 = \gamma_2 = \gamma$ (if available). The same conditions for more general algorithms can be found in our complete paper.

Theorem 3 (EG) *For generalized EG with $\alpha_1 = \alpha_2 = \alpha$ and $\gamma = \beta/\alpha$, linear convergence is achieved iff for any singular value σ of E, we have $\alpha^2\sigma^2 + (\beta\sigma^2 - 1)^2 < 1$ for the Jacobi update, and $0 < \beta\sigma^2 < 2$ and $|\alpha\sigma| < 2 - \beta\sigma^2$ for the GS update. If $2\beta + \alpha^2 < 2/\sigma_1^2$, the convergence region of GS updates **strictly** include that of Jacobi updates.*

Theorem 4 (OGD) *For generalized OGD with $\alpha_1 = \alpha_2 = \alpha$, linear convergence is achieved iff for any singular value σ of E, we have: $0 < \beta\sigma < 1$, $\beta < \alpha < \beta\frac{3-\beta^2\sigma^2}{1+\beta^2\sigma^2}$ for the Jacobi update, and $|\alpha + \beta|\sigma < 2$, $|1 + \alpha\beta\sigma^2| > 1 + \beta^2\sigma^2$ for the GS update. The convergence region of GS updates **strictly** include that of Jacobi updates.*

Theorem 5 (momentum) *For generalized momentum with $\alpha_1 = \alpha_2 = \alpha$, the Jacobi update never converges, while the GS update with $\beta_1 = \beta_2 = \beta$ converges iff for any singular value σ of E, we have $-1 < \beta < 0$, $|\alpha\sigma| < 2(1 + \beta)$. If $\beta_2 = 0$, the exact condition is $-1 < \beta_1 < 0$ and $0 < \alpha\sigma_1 < 2\sqrt{1 + \beta_1}$.*

Prior to our work, only sufficient conditions for linear convergence are given for the usual EG and OGD; see Section 2 above. For the momentum method, our result improves upon [11] where the authors only considered specific cases of parameters. For example, they only considered $\beta \geq -1/16$ for Jacobi momentum, and $\beta_1 = -1/2$, $\beta_2 = 0$ for GS momentum. Our Theorem 5 gives a more complete picture. (For an even more general result please refer to our ICLR paper.)

In the theorems above, we use the term "convergence region" to denote a set of the parameters (α, β or γ) where the algorithm converges. Our result shares similarity with the Stein–Rosenberg theorem [28], which only applies to solving linear systems with non-negative matrices. In this sense, our results extend the Stein–Rosenberg theorem to cover nontrivial bilinear games.

4 Optimal rates

In this section we study the optimal convergence rates of EG and OGD. We define the exponent of linear convergence as $r = \lim_{t\to\infty} ||\mathbf{z}^{(t)}||/||\mathbf{z}^{(t-1)}||$. For ease of presentation we fix $\alpha_1 = \alpha_2 = \alpha > 0$ and we use r_* to denote the optimal rate (w.r.t. the parameters α, β, γ). In Theorem 7, the exact formula β_* in Jacobi OGD, as well as more relevant results, can be found in our full paper.

Theorem 6 (EG optimal) *Both Jacobi and GS EG achieve the optimal exponent of linear convergence $r_* = (\kappa^2 - 1)/(\kappa^2 + 1)$ at $\alpha \to 0$ and $\beta_1 = \beta_2 = 2/(\sigma_1^2 + \sigma_n^2)$. As $\kappa \to \infty$, $r_* \to 1 - 2/\kappa^2$.*

Theorem 7 (OGD optimal) *For Jacobi OGD with $\beta_1 = \beta_2 = \beta$, to achieve the optimal linear convergence, we must have $\alpha \leq 2\beta$. At $\beta = \alpha/2 = \beta_*$, $r_* \sim 1 - 1/(6\kappa^2)$ at large κ. For GS OGD with $\beta_2 = 0$, $r_* = \sqrt{(\kappa^2 - 1)/(\kappa^2 + 1)} \sim 1 - 1/\kappa^2$, at $\alpha = \sqrt{2}/\sigma_1$ and $\beta_1 = \sqrt{2}\sigma_1/(\sigma_1^2 + \sigma_n^2)$.*

5 Experiments

Bilinear game We experiment on a bilinear game and choose the optimal parameters as suggested in Theorem 6 and 7. The results, shown in Figure 1, agree with our theory.

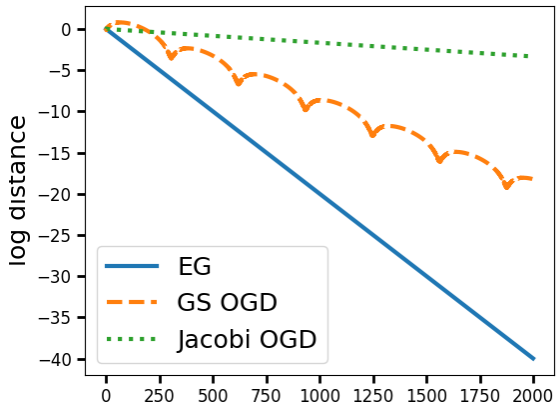


Figure 1: Linear convergence of optimal EG, Jacobi OGD, Gauss–Seidel OGD in a bilinear game

Wasserstein GAN As in [5], we consider a WGAN [1] that learns the mean of a Gaussian: with $s(x)$ the sigmoid function. Near the saddle point $(\theta^*, \phi^*) = (0, v)$ the min-max optimization can be treated as a bilinear game. Since we are doing stochastic versions of the algorithms, we should not expect they will converge exactly to a saddle point. Instead, convergence to a neighborhood is good enough.

With GS updates, we find that Adam [15] diverges, SGD goes around a limit cycle, and EG converges, as shown in the left panel of Figure 2. Our next experiment shows that generalized algorithms may have an advantage over traditional ones. Inspired by Theorem 6, we compare the convergence of two EGs with the same parameter $\beta = \alpha\gamma$, and find that with scaling (decreasing α), EG converges faster to a neighborhood of the saddle point with less oscillation, as shown in the right panel of Figure 2. Note that we always use the squared distance as a measure of convergence.

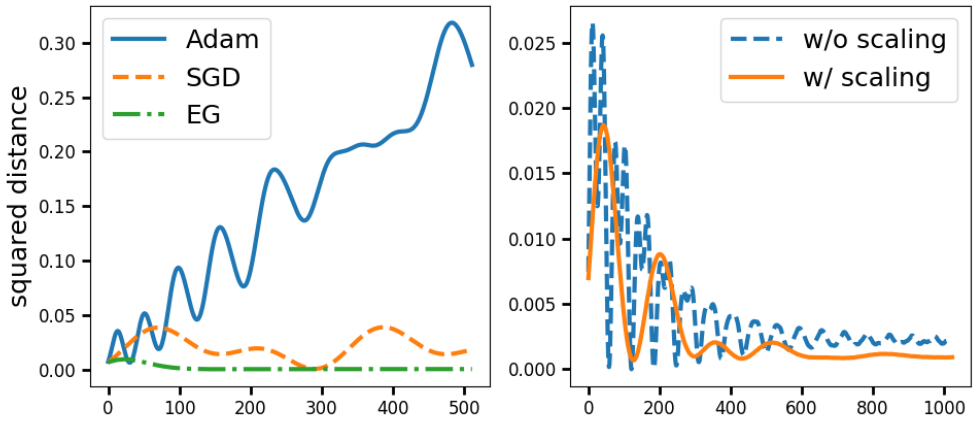


Figure 2: Left: comparison among gradient algorithms; Right: the scaling effect of EG

Finally, we compare Jacobi updates with GS updates. In Figure 3, GS updates converge even when the corresponding Jacobi updates do not.

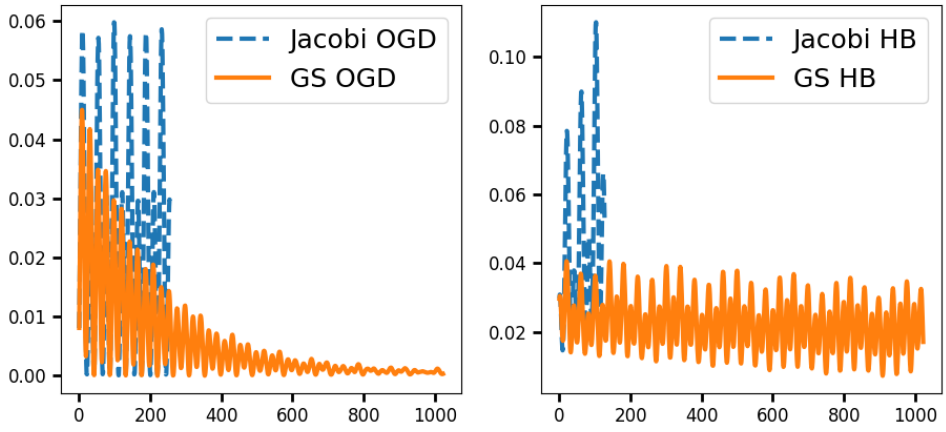


Figure 3: Jacobi vs. GS updates. Left: OGD with $\alpha = 0.2$, $\beta_1 = 0.1$, $\beta_2 = 0$; Right: Momentum with $\alpha = 0.08$, $\beta = -0.1$. We plot only a few epochs for Jacobi updates if they do not converge

6 Conclusions

In this paper, we study convergence of gradient algorithms on bilinear games. Surprisingly, even such a simple game could provide us with great insights for practice. The lessons we have learned are: alternating updates are often more stable than simultaneous updates; by generalizing existing algorithms we can achieve faster convergence rates. We provide guidance for choosing hyper-parameters in bilinear games which could potentially generalize to GAN training.

Acknowledgement

We thank Argyrios Deligkas and Sarath Pattathil for pointing out additional references. This work is supported by NSERC.

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In International Conference on Machine Learning, 2017.

[2] K. J. Arrow, L. Hurwicz, and H. Uzawa. Studies in linear and non-linear programming. Stanford University Press, 1958.

[3] R. E. Bruck. On the weak convergence of an ergodic iteration for the solution of variational inequalities for monotone operators in Hilbert space. Journal of Mathematical Analysis and Applications, 61(1):159–164, 1977.

[4] X. Chen, X. Deng, and S.-H. Teng. Settling the complexity of computing two-player Nash equilibria. Journal of the ACM, 56(3):14, 2009.

[5] C. Daskalakis, A. Ilyas, V. Syrgkanis, and H. Zeng. Training GANs with optimism. In International Conference on Learning Representations, 2018.

[6] A. Deligkas, J. Fearnley, R. Savani, and P. Spirakis. Computing approximate Nash equilibria in polymatrix games. Algorithmica, 77(2):487–514, 2017.

[7] V. F. Dem'yanov and A. B. Pevnyi. Numerical methods for finding saddle points. USSR Computational Mathematics and Mathematical Physics, 12(5):11–52, 1972.

[8] S. S. Du, J. Chen, L. Li, L. Xiao, and D. Zhou. Stochastic variance reduction methods for policy evaluation. In International Conference on Machine Learning, pages 1049–1058, 2017.

[9] Y. Freund and R. E. Schapire. Adaptive game playing using multiplicative weights. Games and Economic Behavior, 29(1-2):79–103, 1999.

[10] G. Gidel, H. Berard, G. Vignoud, P. Vincent, and S. Lacoste-Julien. A variational inequality perspective on generative adversarial networks. In International Conference on Learning Representations, 2019.

[11] G. Gidel, R. A. Hemmat, M. Pezeshki, G. Huang, R. Lepriol, S. Lacoste-Julien, and I. Mitliagkas. Negative momentum for improved game dynamics. In AISTATS, 2019.

[12] I. Gohberg, P. Lancaster, and L. Rodman. Matrix polynomials. Academic Press, 1982.

[13] E. G. Gol'shtein. A generalized gradient method for finding saddlepoints. Ekonomika i matematicheskie metody, 8(4):569–579, 1972.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Advances in Neural Information Processing Systems, pages 2672–2680, 2014.

[15] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In International Conference on Learning Representations, 2015.

[16] G. M. Korpelevich. The extragradient method for finding saddle points and other problems. Matecon, 12:747–756, 1976.

[17] T. Liang and J. Stokes. Interaction matters: A note on non-asymptotic local convergence of generative adversarial networks. In AISTATS, 2019.

[18] P. L. Lions. Une méthode itérative de résolution d'une inéquation variationnelle. Israel Journal of Mathematics, 31(2):204–208, 1978.

[19] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In International Conference on Learning Representations, 2018.

[20] L. Mescheder, S. Nowozin, and A. Geiger. The numerics of GANs. In Advances in Neural Information Processing Systems, pages 1825–1835, 2017.

[21] A. Mokhtari, A. Ozdaglar, and S. Pattathil. A unified analysis of extra-gradient and optimistic gradient methods for saddle point problems: Proximal point approach. arXiv preprint arXiv:1901.08511, 2019.

[22] A. S. Nemirovski and D. B. Yudin. Problem complexity and method efficiency in optimization. Wiley, 1983.

[23] Y. Nesterov. A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$. Doklady Akademii Nauk, 269:543–547, 1983.

[24] Christos H Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. Journal of Computer and system Sciences, 48(3):498–532, 1994.

[25] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics, 4(5):1–17, 1964.

[26] R. T. Rockafellar. Monotone operators and the proximal point algorithm. SIAM journal on control and optimization, 14(5):877–898, 1976.

[27] I. Schur. Über Potenzreihen, die im Innern des Einheitskreises beschränkt sind. Journal für die reine und angewandte Mathematik, 147:205–232, 1917.

[28] P. Stein and R. L. Rosenberg. On the solution of linear simultaneous equations by iteration. Journal of the London Mathematical Society, 1(2):111–118, 1948.

[29] P. Tseng. On linear convergence of iterative methods for the variational inequality problem. Journal of Computational and Applied Mathematics, 60(1-2):237–252, 1995.

[30] Guojun Zhang and Yaoliang Yu. Convergence behaviour of some gradient-based methods on bilinear games. In International Conference on Learning Representations, 2020.

7 Random bias initialization improves quantized training

Xinlin Li [a]
Vahid Partovi Nia [b]

[a] *Huawei Noah's Ark Lab, Montréal (Québec), Canada, H3N 1X9*

[b] *GERAD, HEC Montréal, Montréal (Québec), Canada, H3T 2A7*

xinlin.li1@huawei.com
vahid.partovinia@huawei.com

April 2020
Les Cahiers du GERAD
G-2020-23-EIW07

Abstract: *Binary neural networks improve computationally efficiency of deep models with a large margin. However, there is still a performance gap between a successful full-precision training and binary training. We bring some insights about why this accuracy drop exists and call for a better understanding of binary network geometry. We start with analyzing full-precision neural networks with ReLU activation and compare it with its binarized version. This comparison suggests to initialize networks with random bias, a counter-intuitive remedy.*

1 Introduction

It is common to use low-bit quantized networks such as Binary Neural Networks (BNNs) [1] to implement deep neural networks on edge devices such as cell phones, smart wearables, etc. BNNs only keep the sign of weights and compute the sign of activations $\{-1, +1\}$ by applying the sign function in the forward pass. In backward propagation, BNN uses Straight-Through-Estimator (STE) to estimate the backward gradient through the sign function and update on full-precision weights. The forward and backward loop of a BNN, therefore, becomes similar to the full-precision neural network with hard hyperbolic tangent *htanh* activation. The htanh function is a piece-wise linear version of the nonlinear hyper-bolic tangent, and is known to be inferior in terms of accuracy compared to ReLU-like activation function. Although the analysis is based on htanh function, this conclusion equally applies to BNNs that use STE, a htanh-like, back propagation scheme. Other saturating activations like hyperbolic tangent and sigmoid commonly applied in recurrent neural networks and attention-based models may benefit from this resolution as well. Among others, [3] recommends an initialization scheme for binary weights but ignores the bias term. [2] utilized automatic search techniques on searching different activation functions. Most top novel activation functions found by the searches have an asymmetric saturating regime, which is similar to ReLU.

2 Full-precision networks

A typical full-precision neural network block can be described by

$$\begin{gathered} x^{i+1} = \mathrm{ReLU}(W^i x^i + b^i) \\ W^i \in \mathbb{R}^{m\times n}, b^i \in \mathbb{R}^m, x^i \in \mathbb{R}^n, x^{i+1} \in \mathbb{R}^m. \end{gathered} \tag{1}$$

Neural networks are trained using the back-propagation algorithm. Back propagation is composed of two components i) forward pass and ii) backward propagation. In the forward pass, the loss function $\mathcal{L}(.)$ is evaluated on the current weights, and in backward propagation, gradients and then weights are updated sequentially.

Assume weight vectors W_j^i have unit norm. It is a reasonable assumption when the network has batch normalization layers in which all neuron responses are normalized, as the magnitude of the weight vectors does not affect the layer output. The j^{th} neuron response in the $(i+1)^{th}$ layer are computed as

$$x_j^{i+1} = \begin{cases} W_j^i x^i + b_j^i & W_j^i x^i + b_j^i > 0 \\ 0 & W_j^i x^i + b_j^i \leq 0 \end{cases} \tag{2}$$

First, the input data points x^i are projected to the j^{th} row vector of the weight matrix. The dot product of W_j^i and x^i are cut by the corresponding bias term b_j^i, i.e. the output x_j^{i+1} is set to zero if the dot product is smaller than the threshold, see Figure 2 (left panel). A hyper-plane whose normal direction defined by W_j^i divides the input space into two parts: i) activated region (non-saturated regime) and ii) non-activated region (saturated regime), see Figure 1. If the data point x^i falls on the positive side of a hyper-plane (activated region), the hyper-plane is activated by x^i. Consequently, x_j^{i+1} is positive. Otherwise, x_j^{i+1} equals zero and remains inactive.

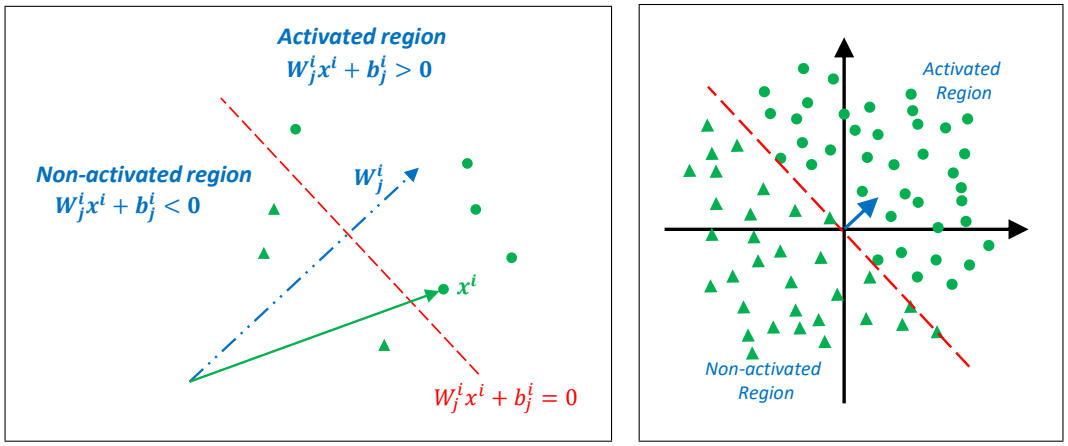


Figure 1: Activated and non-activated regions of ReLU (left panel). Activated region of ReLU at initialization (right panel)

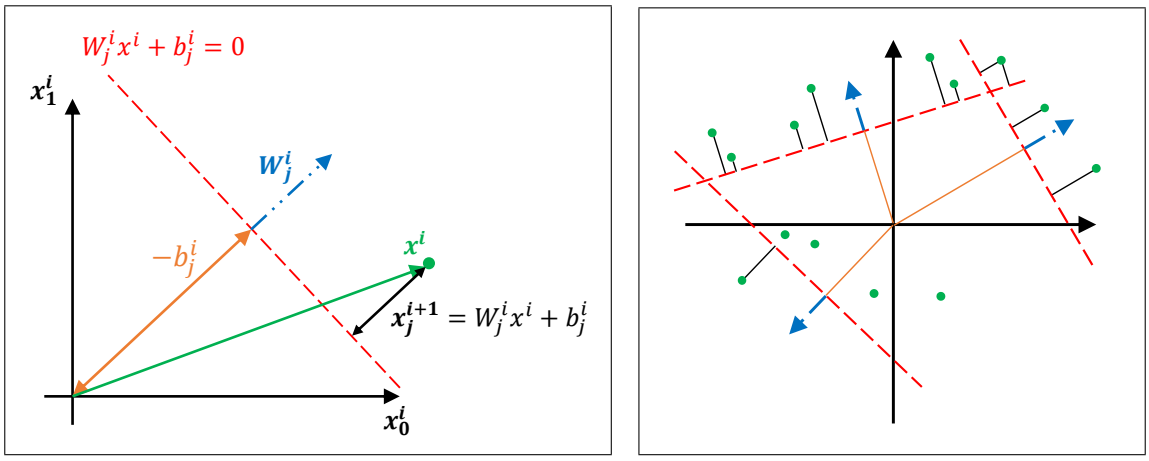


Figure 2: Geometric behavior of ReLu during forward pass, trained hyperplanes (left panel) and their geometry (right panel)

The weight matrix W^i of size $m \times n$ and the bias vector b^i of size $m \times 1$ define m hyper-planes in the n-dimensional input space, see Figure 2 (right panel).

During backward propagation, the backward gradient update on W_j^i and x^i are computed using

$$\begin{cases} \frac{d\mathcal{L}}{dW_j^i} = \frac{d\mathcal{L}}{dx_j^{i+1}} * \frac{dx_j^{i+1}}{dW_j^i} \\ \frac{d\mathcal{L}}{db_j^i} = \frac{d\mathcal{L}}{dx_j^{i+1}} * \frac{dx_j^{i+1}}{db_j^i} \\ \frac{d\mathcal{L}}{dx^i} = \frac{d\mathcal{L}}{dx_j^{i+1}} * \frac{dx_j^{i+1}}{dx^i} \end{cases} \tag{3}$$

For the case of ReLU activation

$$\frac{dx_j^{i+1}}{dW_j^i} = \begin{cases} x^i & W_j^i x^i + b_j^i > 0 \\ 0 & W_j^i x^i + b_j^i \leq 0 \end{cases} \tag{4}$$

$$\frac{dx_j^{i+1}}{db_j^i} = \begin{cases} 1 & W_j^i x^i + b_j^i > 0 \\ 0 & W_j^i x^i + b_j^i \leq 0 \end{cases} \tag{5}$$

$$\frac{dx_j^{i+1}}{dx^i} = \begin{cases} W_j^i & W_j^i x^i + b_j^i > 0 \\ 0 & W_j^i x^i + b_j^i \leq 0 \end{cases} \tag{6}$$

The activation function only allows the gradients from data point on the activated region to backward propagate and update the hyper-plane (4).

From the hyper-plane analysis, we realize that ReLU activation has three ideal properties i) the diversity of activated regions at initialization, ii) The equality of data points at initialization, iii) The equality of hyper-planes at initialization which we discuss each property in more details later. These may explain why ReLU activation outperforms the traditional Hyperbolic tangent or sigmoid activations. To argue each property, let us suppose that the distribution of the dot products is zero-centered. This assumption is automatically preserved in neural networks with batch normalization layer.

i) Region diversity: the activated regions of hyper-planes solely depend on the direction of the weight vector, which is randomly initialized. This allows different hyper-planes to learn from a different subset of data points, and ultimately diversifies the backward gradient signal. ii) Data equality: an arbitrary data point x^i, is located on activated regions of approximately half of the total hyper-planes in a layer. In other words, the backward gradients from all data points can pass through the approximately same amount of activation function, update hyper-planes, and propagate the gradient. iii) Hyperplane equality: an arbitrary hyper-plane W_j^i, is affected by the backward gradients from approximately 50% of the total data points. All hyper-planes on average receive the same amount of backward gradients. Hyper-plane equality speeds up the convergence and facilitates model optimization, see Figure 1 (right panel).

The performance gap between ReLU activation and htanh activation is caused by their different activated region distribution, see Figure 3. Clearly, htanh activation is not as good as ReLU in defining balanced and fair activated regions. However, we analyze each property for htanh as well.

i) Region diversity: activated regions of htanh are not as diverse as ReLU. Activated regions of htanh cover only the area close to the origin. Assuming Gaussian data, this is a dense area that the majority of data points are located in.

ii) Data equality: data points are not treated fairly htanh activation function. Data points that closer to the origin can activate more hyper-planes than the data points far from the origin. If the magnitude of a data point x^i is small enough, it can activate all hyper-planes in the same layer, see the deep-red region of Figure 3 (right panel). As a consequence, in backward gradients, few data instances affect all hyper-planes. In other words, the backward gradients from a part of the training data have a larger impact on model than others. This imbalance ultimately affects model generalization problem since the model training focuses only on a subset of the training data points close to the origin.

iii) Hyperplane equality: The initial activated regions should cover a similar-sized subset of the training data points overall, and this property is shared in both ReLU and htanh activations. Similar analysis also applies to other activation functions with the zero-centered activated region, like sigmoid or tanh.

3 Training acceleration

Here we proposed a simple initialization strategy to alleviate the data inequality issue and improve activated region diversity for the htanh activation relying on our geometric insight described earlier. We argue bias initialization with a uniform distribution between $[-\lambda, \lambda]$, where λ is a hyper-parameter is a quick remedy. With random bias initialization, the data points that far from the origin can activate more hyper-planes. If $\lambda > \max(\|x\|) + 1$, all data points activate approximately the same number of hyper-planes during backward propagation, so data equality can be achieved. Also, with the diverse initial activated region, different hyper-planes learn from different subset of training data points.

However, this initialization strategy comes with a drawback. Hyper-plane equality no longer holds when the biases are not set to zero. Hyper-planes with larger initial bias have less activated data.

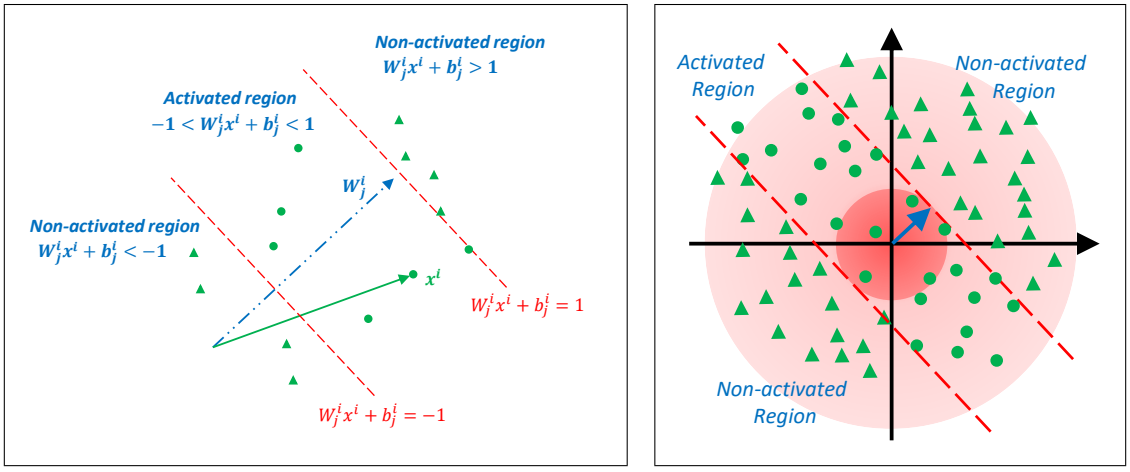


Figure 3: Activated region and non-activated region of htanh activation function(left panel). Activated region of Hard Tanh at initialization (right panel)

Therefore, choosing the optimal value of λ is a trade-off between the hyper-plane equality and the data equality. Experiments below shows that the validation curve becomes unsteady if λ value set to too high. Empirically, with a batch normalization layer, $\lambda \approx 2$ provide a good initial estimate. In this case, the activated regions covering from -3 to $+3$, so it allows the gradients from almost all data points to propagate. Our experiments shows small λ also helps to improve the performance of ResNet architecture.

4 Numerical Results

The proposed bias initialization method is evaluated on the CIFAR-10. The network architectures are based on the original implementation of the BNN [1]. We choose the VGG-7 architecture and the ResNet architecture.

The VGG-7 architecture, is a simple and over-parameterized model for CIFAR 10. This is an ideal architecture to compare the performance between different activations. Figure 4 confirms that the random bias initialization strategy helps to reduce the performance gap between htanh and ReLU activation. A similar effect is observed for ResNet type architectures.

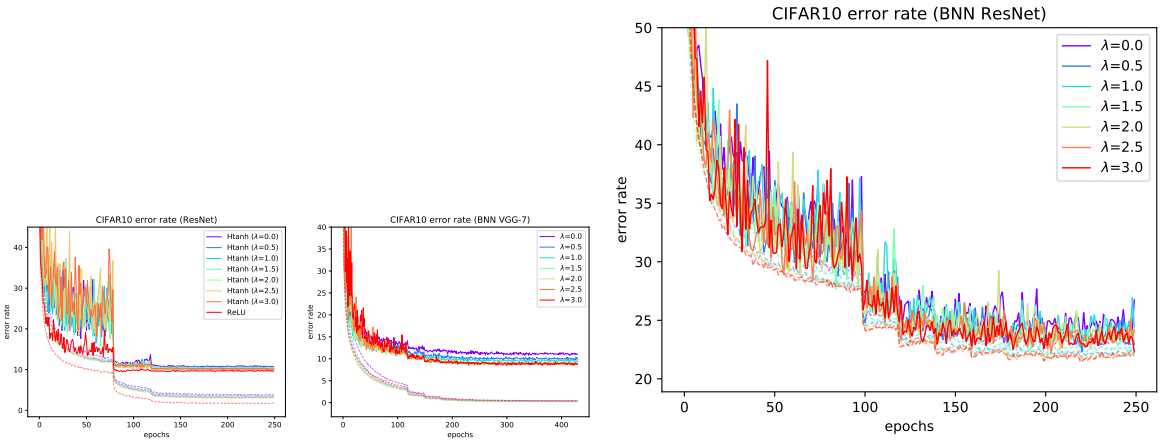


Figure 4: Training of full-precision ResNet architecture (top left panel) Binary VGG-7 architecture (top right), and Binary ResNet (bottom panel)

We also tested the proposed bias initialization on the ResNet-like architecture. The results are depicted in Figure 4 re-assures that bias initialization improves htanh and pushes it toward ReLU accuracy, see Table 1.

Table 1: Validation error rate % for full-precision training, $\lambda = 0$ coincides with common deterministic initialization

Activations	VGG-7	ResNet
ReLU (Baseline)	6.98	9.45
htanh	10.91	10.63
htanh (λ=0.5)	9.99	9.87
htanh (λ=1.0)	9.15	10.47
htanh (λ=1.5)	8.36	10.13
htanh (λ=2.0)	7.98	10.23
htanh (λ=2.5)	**7.83**	**9.84**

Binary training that use STE is similar to htanh activation. We expect to observe a similar effect in BNN training with STE gradient approximator. The validation error rate is summarized in Table 2. In the Binary VGG-7 experiments, we reduced the accuracy gap between full-precision network with ReLU activation and BNN from 4% to 1.5%. The bias initialization strategy is effective to close the gap on binary ResNet architecture by almost 1%, even while the full-precision model even under-fits on CIFAR10 data.

Table 2: Validation error rate % for Binary training, $\lambda = 0$ coincides with common deterministic initialization

λ	Binary VGG-7	Binary ResNet
0.0	10.77	23.11
0.5	9.57	22.31
1.0	9.17	22.83
1.5	8.57	22.56
2.0	8.56	22.47
2.5	8.53	**22.19**
3.0	**8.48**	22.30

5 Conclusion

We analyzed different geometric behaviour of ReLU activated and hard tanh activated full-precision neural network. The analysis implies the superior performance of ReLU activation may come from its three preferred geometric properties, region diversity, data equality and hyper-plane equality. However, a theoretical investigation is required to prove this claim. We proposed to use random bias initialization in hard tanh activated neural network to micmic the geometric properties of ReLU. The same analysis can also apply to binary neural networks with Straight-Through-Estimator back-propagation scheme. Our numerical experiments confirm our geometric intuition. The CIFAR10 experiments show the proposed random bias initialization reduces the performance gap between ReLU activation and hard tanh activation on ResNet and VGG architectures. This initialization strategy improves the binary neural network performance as well.

6 Acknowledgement

We would like to thank Huawei CBG Software Shanghai colleagues, especially Mohan Liu and Li Zhou for their fruitful technical discussions. We also thank Yanhui Geng and Jin Tang for their support throughoutthe project, and Masoud Asgharian for reviewing the manuscript.

References

[1] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1. 2016.

[2] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. arXiv preprint arXiv:1710.05941, 2017.

[3] Eyyüb Sari, Mouloud Belbahri, and Vahid Partovi Nia. How does batch normalization help binary training? arXiv preprint arXiv:1909.09139v2, 2019.

8 Importance of data loading pipeline in training deep neural networks

Mahdi Zolnouri [a]
Xinlin Li [a]
Vahid Partovi Nia [a,b]

[a] *Huawei Noah's Ark Lab, Montréal (Québec), Canada, H3N 1X9*

[b] *GERAD, HEC Montréal, Montréal (Québec), Canada, H3T 2A7*

mahdi.zolnouri@huawei.com
xinlin.li1@huawei.com
vahid.partovinia@huawei.com

April 2020
Les Cahiers du GERAD
G-2020-23-EIW08

Abstract: *Training large-scale deep neural networks is a long, time-consuming operation, often requiring many GPUs to accelerate. In large models, the time spent loading data takes a significant portion of model training time. As GPU servers are typically expensive, tricks that can save training time are valuable. Slow training is observed especially on real-world applications where exhaustive data augmentation operations are required. Data augmentation techniques include: padding, rotation, adding noise, down sampling, up sampling, etc. These additional operations increase the need to build an efficient data loading pipeline, and to explore existing tools to speed up training time. We focus on the comparison of two main tools designed for this task, namely binary data format to accelerate* data reading*, and NVIDIA DALI to accelerate* data augmentation*. Our study shows improvement on the order of* 20% *to* 40% *if such dedicated tools are used.*

1 Introduction

Deep neural networks have achieved great successes in various domains such as computer vision [3, 2], natural language processing [10, 9], and speech recognition [17] among others. This is a result of deeper and wider models, which allow modeling large and complex data. As computing hardware has improved, larger data sets are analyzed. It appears that processing power always falls behind data volume and model size. In order to make the training process more efficient, several fields are developing new techniques such as providing dedicated tools to accelerate training and inference, as well as neural model compression to deploy a simpler model with comparable accuracy but fewer operations.

Training acceleration is a difficult task. Let's understand the core of the problem using an a very simple neural network, e.g. logistic regression. Suppose N pairs of input features of dimension d, say $\mathbf{x}_i$ and binary output data, say y_i are available $(\mathbf{x}_i, y_i), i = 1, \ldots, N$. A logistic regression model is equivalent to a fully-connected network with a single hidden layer and a single neuron. As the data size gets bigger in terms of N and d, training requires more computation.

A training process optimizes a loss function, here

$$\begin{aligned}\mathcal{L}(\mathbf{w}) = -\sum_{i=1}^{n} & y_i \log \sigma(\mathbf{x}_i^\top \mathbf{w}) \\ & + (1 - y_i) \log(1 - \sigma(\mathbf{x}_i^\top \mathbf{w})),\end{aligned} \tag{1}$$

where

$$\sigma(x) = \{1 + \exp(-x)\}^{-1}$$

is the sigmoid activation. For the case of logistic regression iterative re-weighted least squares is often used to optimize $\mathcal{L}$, which is equivalent to Newton's method. Newton's method starts from an initial estimate $\mathbf{w}_0$ and updates

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{H}^{-1} \nabla \mathcal{L}(\mathbf{w}) \mid_{\mathbf{w}=\mathbf{w}_t} \tag{2}$$

where $\mathbf{H}$ is the $d \times d$ Hessian of $\mathcal{L}$ and $\nabla \mathcal{L} = \sum_{i=1}^{n} \nabla \mathcal{L}_i(\mathbf{w})$ is the sum of individual gradients, each of length d. Computing the local approximation of $\mathbf{H} = \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^\top$ is of $\mathcal{O}(nd^2)$ and factorizing it is of $\mathcal{O}(nd^3)$, if not impossible.

Increasing the number of data points $N \to \infty$ theoretically makes the optimization easier, because $\mathcal{L}(\mathbf{w})$ has more curvature as N increases, a *blessing*. However, computation of $\mathcal{L}(\mathbf{w})$, $\nabla \mathcal{L}(\mathbf{w})$, and $\mathbf{H}$ becomes a *curse* since all of these quantities are in the form of a sum and their computation may lead to memory overflow. Optimization using Newton's method becomes increasingly hard with large feature size d. Large features are common in almost all machine learning challenges. The remedy for large N is to break computations into smaller sub-operations. The remedy for large d is to switch from a second-order approximation, i.e., Newton's method, to a first-order approximation, i.e., the gradient method.

Computing partial sums is a simple way to overcome the large N issue, so that each partial sum remains within the memory resource constraints. Then the final quantity is computed by summing the partial sums, perhaps with a proper re-scaling. The idea of partial sum is somehow a re-shape of the mini batch training approach.

In neural networks with a large d and n, numerical optimization is simplified to gradient descent in which the hessian $\mathbf{H}$ is replaced by the identity matrix with a positive scalar learning rate

$$\mathbf{H} = \frac{1}{\eta}\mathbf{I}, \eta > 0$$

so the weight update is simplified to

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \nabla \mathcal{L}(\mathbf{w}) \mid_{\mathbf{w}=\mathbf{w}_t} . \tag{3}$$

Furthermore, to benefit from parallel computation, N data points are arranged in n random mini batches, each of size B, i.e. $N = nB$. Each batch has its own gradient

$$\nabla \mathcal{L}_b(\mathbf{w}) = \sum_{i=1}^{B} \nabla \mathcal{L}_{bi}(\mathbf{w}),$$

which is equivalent to scaling η by n, on average. This allows computations to be run in parallel for each batch [4, 16].

Even if computations are run in parallel, all data still needs to be fed to the optimizer in several rounds of *epochs*, similar to Newton's method. Therefore, investing in an efficient data loading pipeline plays an important role in training speed [18].

The rest of the paper focuses on clarifying the benefit of a dedicated tool such as DALI[1] for managing data loading implemented by NVIDIA in PyTorch, while a using a convenient data reading format such as Hierarchical Data Format 5 (HDF5) [14] or TensorFlow Record [1] to accelerate file reading.

Data loading is a crucial part of model training in neural networks. It begins by reading the data from a secondary memory storage, such as *Solid State Drive*, then caches it into a primary memory storage, such as *Random Access Memory*. This data transfer includes extra operations like data augmentation to feed the data to the model. See Figure 1.

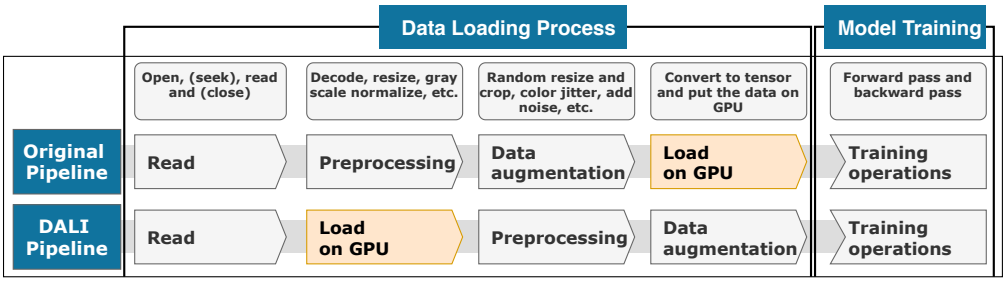


Figure 1: Batch training sequence performed on a GPU with and without DALI

There are two main issues in the data loading process i) reading data directly from files is inefficient ii) resource allocation for extra operations on data overloads the CPU.

Reading data from individual files directly is slow. This happens when the entire data set is not cached in the local memory. Every single file is opened, is read, and is closed sequentially. These sequential operations add considerable overhead to the file retrieval time. One solution to prevent

[1] https://github.com/NVIDIA/DALI

this overhead is to use the Hierarchical Data Format version 5 (HDF5) [14], which has an open-source library to store, manipulate, and manage the large data set. HDF5 format stores multiple data sets in one file as a multidimensional array of binary data. It also groups storage layout by storing data in fixed-size chunks on disk. Another alternative to HDF5 is the TensorFlow Record (TFRecord) [1], which uses a sequence of binary strings to store data. It allows large data sets to be sequentially loaded to the local memory.

The whole process of data loading is managed by CPU, which can create a bottleneck for model training. This bottleneck happens normally in the case of multi-node-multi-GPU training, as loading batches of data takes more time than forward-backward propagation. To prevent the CPU bottleneck issue, NVIDIA Data Loading Library (DALI) helps by sharing some data loading tasks between the CPU and GPU, to prevent the CPU bottleneck issue. Figure 2 shows how employing this library improves the data loading process considerably. DALI is a collection of highly optimized building blocks and execution engines which provides a full data accelerated pipeline: from reading the data to preparing for training and inference.

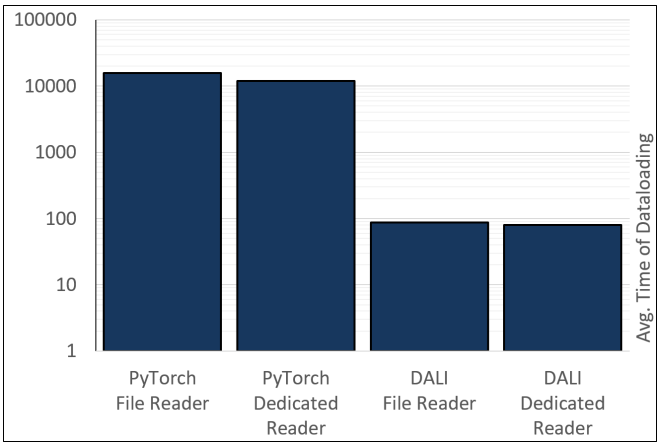


Figure 2: Logarithmic scale of average time of data loading (in micro second) for ResNet-50 [12] on ImageNet data set with batch-size $B = 256$

2 Data format

In order to explore the effect that data format and resource allocation has on data loading performance, we present four pipelines for data loading: with/without dedicated reader, and with/without resource allocation. We used PyTorch version 1.2.0 to implement these pipelines on a computer vision task.

The PyTorch *file reader* pipeline is the PyTorch *DataLoader* class which combines *data set* object with a *sampler* object, to provide a single or multi-process iterators over the data set. This pipeline reads data from individual JPEG files on the storage and uses the PyTorch *Transforms* class to chain several image transformation operations together. This prepares the data for training by doing the data augmentation. By default, all these operations are directed to the CPU.

The *dedicated reader* pipeline is also based on the PyTorch *DATALOADER* class. This pipeline stores data differently, i.e. instead of reading data from individual JPEG files, the entire train and validation data sets are stored as two HDF5 files.

3 Resource allocation

This pipeline uses the NVIDIA DALI library to read data from JPEG files, process and then feed GPUs for training. In the DALI pipeline, the data loading process is shared between CPU and GPU. This means that all operations on data, such as resizing, cropping and data augmentation can be run

on CPU, GPU, or a mix of both. To measure the effect of file formatting, data file retrieval has been done in two cases: reading directly from JPEG files and using a dedicated file reader to read data from TFRecord data set.

4 Training time improvement

We summarize our experiments using four configurations, i.e., with/without a dedicated file reader, and with/without DALI. In order to compare these pipelines, we performed several experiments in two use cases: with few or with extensive *data augmentation* operations. By few operations, we mean resize with random crop and random horizontal flip operations and by extensive, we mean resize with random crop operation, random horizontal flip operation and random adjustment of the brightness, contrast and saturation of an image. Simple data augmentation is applied in most deep learning prototypes, while extensive data augmentation is very common in industry to ensure model robustness in real products. Our experiments are run twice, once on small subset of InsightFace [6, 7, 8, 11] and once on large ImageNet data set [5].

Figure 3 shows that a dedicated data reader is enough to improve epoch time, but only if data is small. However, the DALI pipeline still is a winner for small data requiring extensive data augmentation. This is because DALI distributes the data augmentation operation between CPU and GPU to avoid CPU overcharge. This can be seen by comparing the top left panel, with the top right panel.

If extensive data preprocessing operations are performed in a larger ImageNet data, CPU workload becomes the bottleneck even with few data augmentation operations. Consequently, epoch time increases in data loading and DALI can avoid the CPU overcharge by performing some of these operations on GPU. Figure 3 (bottom right) shows that DALI improves data loading from 2200 seconds to

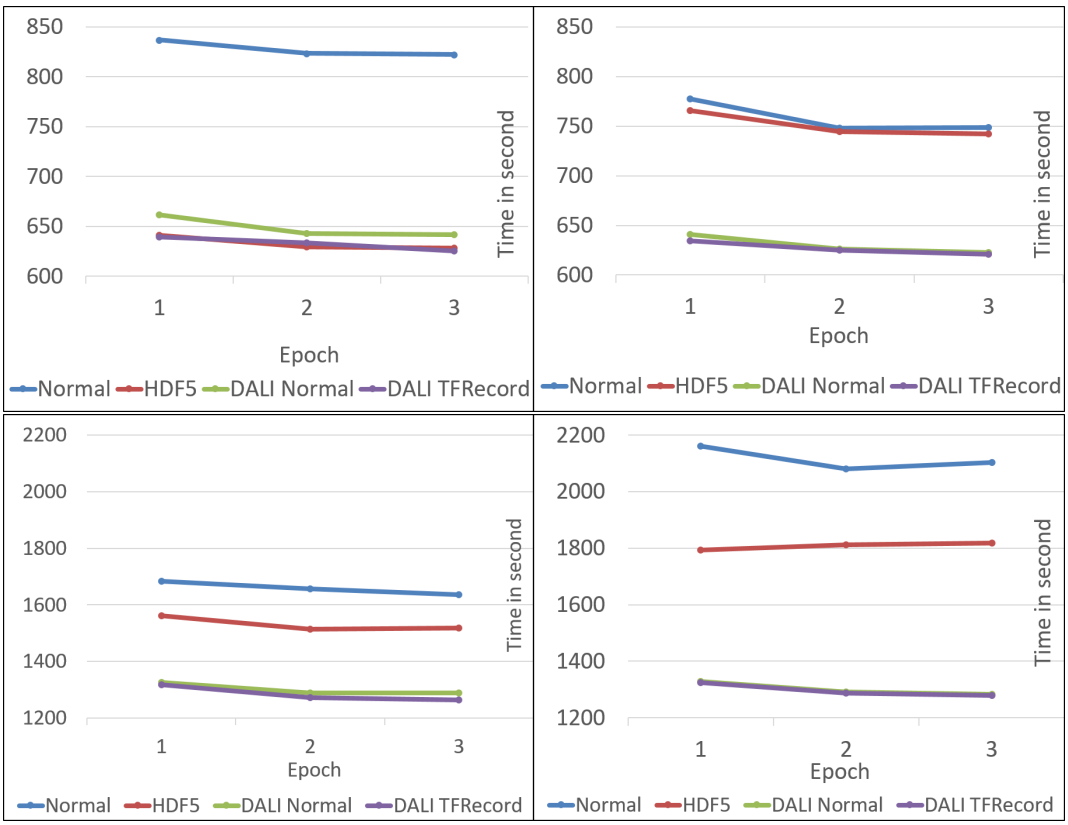


Figure 3: Epoch time for the small subset of InsightFace (top panels) and large ImageNet data (bottom panels), while few data processing operations are performed (left panels) and while extensive preprocessing operations are performed (right panel). The experiments in top panels were performed on 4 GPUs (NVIDIA TITAN 12 GB memory) and in bottom panels on 8 GPUs (NVIDIA TESLA V100 32GB memory)

1300 seconds, to gain performance benefit of about 40%. Using only a dedicated reader without DALI improves the training time from 1800 seconds to 1300 seconds, giving a performance improvement of 30%. This effect is also visible in Figure 4 on a single epoch time. DALI fuses multiple operations such as cropping and normalizing and run it on one GPU CUDA kernel. This speeds up data augmentation process by reducing the number of memory access.

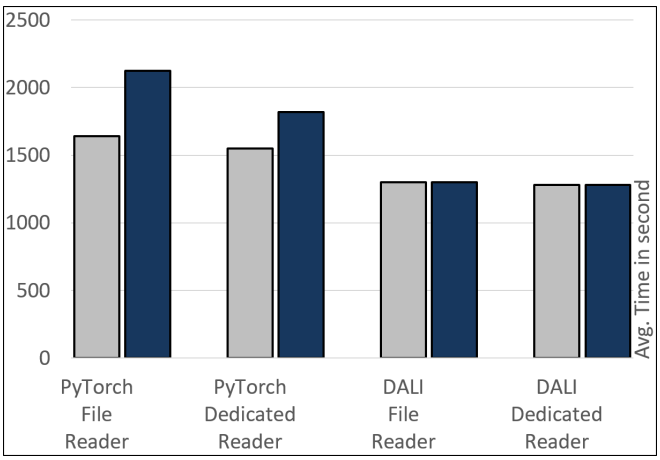


Figure 4: Time comparison for few (gray) and extensive (dark) data augmentations

5 Data loading improvement

Let's move from epoch time to data loading time by removing forward and backward pass from training time, see Figure 1. Data loading contributes about 40% to the epoch time, see Figure 5.

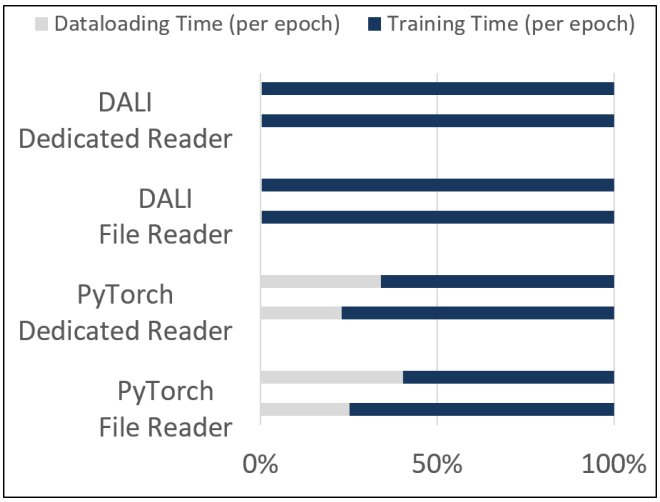


Figure 5: Percentage of data loading time (gray) added on top of training time (dark) in a single training epoch. Each data loader test case has two bar charts for showing few and extensive data augmentations

Figure 6 confirms that the DALI pipeline considerably improves data loading by a factor of 100×. However, for very large models, the GPU is only required to perform forward and backward passes, so loading the CPU for data is wiser. Therefore, it is important to keep the CPU and GPU load well balanced through DALI load option. Figure 7 confirms the same message when training time is stacked on data loading time to measure the epoch time overall.

In Figure 7 data loading is stacked on training to reflect the epoch time overall. There is a small difference between the common data loader pipeline and NVIDIA DALI data loader pipeline. Using

the GPU resources for data loading may slow down the overall model training time if it fails to balance CPU and GPU load.

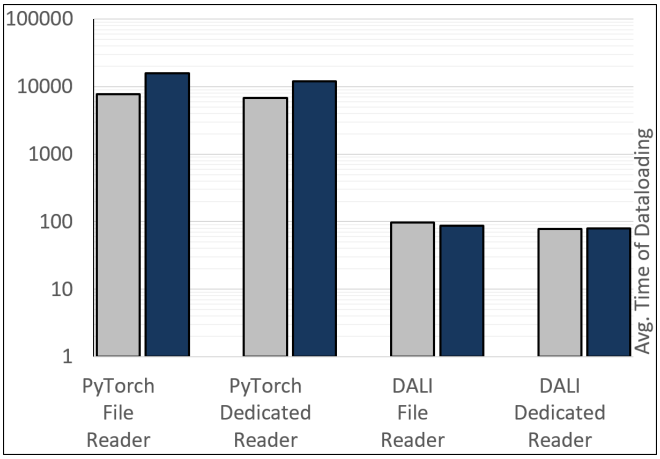


Figure 6: Logarithmic scale plot of average data loading time (per epoch), four data loader pipelines: with/without DALI, and with/without a dedicated reader. Time for few (gray) and extensive (dark) data augmentations

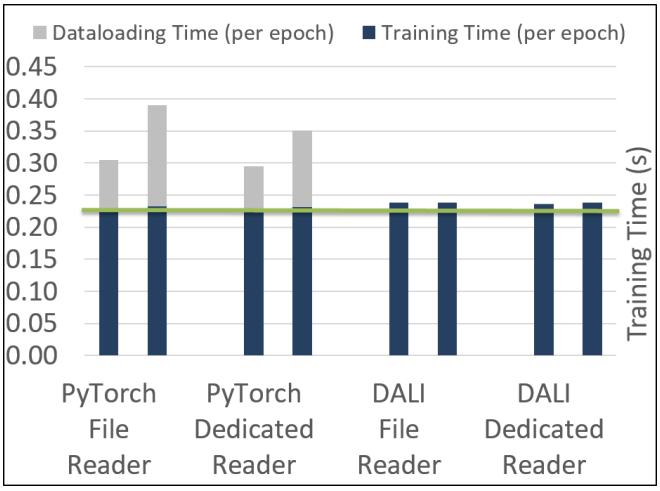


Figure 7: Training time on four data loader pipelines: with/without DALI, and with/without a dedicated reader. Each data loader pipeline has two bar charts for showing few and extensive data augmentations

By default, DALI uses the first GPU slot to perform data loading process. However, the NVIDIA APEX library uses multiple GPUs for this task. This flexibility becomes increasingly important for large models while all data and models cannot be loaded into a single GPU and multi-GPU operation becomes a necessity.

6 Conclusion

NVIDIA DALI provides an effective alternative to common data loading process. It provides a full pipeline of optimizations including data readers and tools to accelerate training and inference. It also enables most data augmentation operations to be performed on GPU and on CPU. In addition, DALI prepared a full pipeline for common data sets like MS-COCO data set [15] as well as provides a reader for TFRecord and CAFFE LMDB data formats [13]. DALI remains extremely flexible by supporting ExternalSource operator so that implementation of unsupported readers such as HDF5 becomes feasible.

Here, we focused on large models, in which DALI GPU improves training time. However, training is faster with DALI CPU for small networks.

Acknowledgment

The authors would like to thank Eyyüb Sari and Vanessa Courville for their assistance in this project. We appreciate fruitful technical discussions with Huawei Cloud Core Shanghai Gang Chi and Pengcheng Tang as well as Jiajin Zhang from Noah's Ark Shenzhen engineering team.

References

[1] Martín Abadi. Tensorflow: learning functions at scale. In Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, pages 1–1, 2016.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE transactions on pattern analysis and machine intelligence, 39(12):2481–2495, 2017.

[3] François Chollet. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1251–1258, 2017.

[4] Andrew Cotter, Ohad Shamir, Nati Srebro, and Karthik Sridharan. Better mini-batch algorithms via accelerated gradient methods. In Advances in neural information processing systems, pages 1647–1655, 2011.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. Ieee, 2009.

[6] Jiankang Deng, Jia Guo, Xue Niannan, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In CVPR, 2019.

[7] Jiankang Deng, Jia Guo, Zhou Yuxiang, Jinke Yu, Irene Kotsia, and Stefanos Zafeiriou. Retinaface: Single-stage dense face localisation in the wild. In arxiv, 2019.

[8] Jiankang Deng, Anastasios Roussos, Grigorios Chrysos, Evangelos Ververas, Irene Kotsia, Jie Shen, and Stefanos Zafeiriou. The menpo benchmark for multi-pose 2d and 3d facial landmark localisation and tracking. IJCV, 2018.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[10] Yoav Goldberg. A primer on neural network models for natural language processing. Journal of Artificial Intelligence Research, 57:345–420, 2016.

[11] Jia Guo, Jiankang Deng, Niannan Xue, and Stefanos Zafeiriou. Stacked dense u-nets with dual transformers for robust face alignment. In BMVC, 2018.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[13] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia, pages 675–678, 2014.

[14] Sandeep Koranne. Hierarchical data format 5: Hdf5. In Handbook of Open Source Tools, pages 191–200. Springer, 2011.

[15] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In European conference on computer vision, pages 740–755. Springer, 2014.

[16] Krishnamurty Muralidhar and Rathindra Sarathy. Data shuffling—a new masking approach for numerical data. Management Science, 52(5):658–670, 2006.

[17] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In IEEE 2011 workshop on automatic speech recognition and understanding, number CONF. IEEE Signal Processing Society, 2011.

[18] Chih-Chieh Yang and Guojing Cong. Accelerating data loading in deep neural network training. arXiv preprint arXiv:1910.01196, 2019.

9 Semi$^+$-supervised learning under sample selection bias

Damoon Robatian [a]
Masoud Asgharian [b]

[a] *GERAD & Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal (Québec) Canada, H3C 3A7*

[b] *Department of Mathematics and Statistics, McGill University, Montréal (Québec) Canada, H3A 2K6*

damoon.robatian@polymtl.ca
masoud.asgharian2@mcgill.ca

April 2020
Les Cahiers du GERAD
G-2020-23-EIW09

Abstract: *In time-to-event data analysis, the main object of interest is the time elapsed between the occurrence of two ordered events, say E_1, E_2. Sampling from the incident population, i.e., subjects who have experienced the incidence of E_1 before being sampled regardless of the occurrence of E_2, is the gold standard in follow-up studies. Yet often in practice, it is more feasible to sample from the prevalent population, i.e., subjects who have already experienced E_1, but not E_2. It is well known that the prevalent sampling design induces sample selection bias. Moreover, time-to-event data are usually subject to censoring which causes partial loss of information on a fraction of the subjects. Here, we discuss the inefficiency of the conventional learning methods due to ignoring sample selection bias and show how this problem can be avoided by properly incorporating the selection bias into the analysis. Arguments are backed by simulation studies.*

1 Introduction

Time-to-event is the output of interest in numerous disciplines spanning epidemiology, economics, econometrics, gerontology, and etc. It is defined as the amount of time elapsed from the occurrence of an initiating event until that of a second event called terminating event. Both events are pre-defined. For example, the initiating event might be birth, onset of a disease, or an aircraft's release, while the terminating event could be retirement, death, or the aircraft's phase-out, respectively. Time-to-event modelling is a ubiquitous problem, an evidence of which is the existence of multiple domains, such as survival analysis, reliability theory, event history analysis, duration modelling, etc., all with similar objectives. As a result, a vast variety of methods have been developed for this purpose. Survival analysis alone hosts a great deal of theory, a big portion of which is related to modelling potential associations between the time-to-event or an individual's survival time and a set of observed measurements for that individual. Naturally, any data-driven inference depends on characteristics of the training data. That is, any quality of the data, potentially affecting the outcome of the analysis, should be properly incorporated in the learning process; otherwise the algorithm's learnability, i.e., the ability to extract relevant information might be influenced negatively. Regarding time-to-event data, there are several points worth considering. One is that data may suffer from multiple types of *incompleteness*, ignoring which may cause serious issues. The gold standard in time-to-event data is to conduct follow-up studies on randomly selected cases from the *incident population*, i.e., subjects who have not experienced the initiating event before the study starts. Logistic or other constraints may, however, preclude the possibility of conducting incident cohort studies. A feasible alternative in such cases is to conduct a *cross-sectional prevalent cohort study* for which one recruits prevalent cases, that is, subjects who have already experienced the initiating event, but not the terminating event. When the interest lies in estimating the lifespan between the initiating and the terminating event, subjects may be followed prospectively either until the terminating event happens or they are lost to follow-up, whichever occurs first. This study design gives rise to two types of incompleteness: First, the response variable, being lifetime, is observed for some subjects while for others we only know that it is greater than some observed period, called censoring time. This type of incompleteness due to censoring is called *right censoring*. Learning from such data for prediction and generalization falls, roughly, into the realm of semi-supervised learning with the difference that there is partial information on subjects whose response is not observed, hence the name semi$^+$-supervised learning. Second, it is well known that prevalent cases have, on average, longer lifespans since longer survivors are more prone to be selected at the recruitment time. As such, a prevalent cohort comprises a biased, and non-random sample of the target population. An important feature of such data is that the response selection bias induces a bias on the feature space's sampling frame. This is so since certain feature values could be preferentially selected into the sample, being linked to the long-term survivors, who themselves are favored by the sampling mechanism. This systematically introduced bias comprises the second type of incompleteness, called sample selection bias. Here, we discuss different challenges in analysing and learning from such data. Particular attention is paid to the case where the chance of being selected into the sample is proportional to the survival time, the so-called length-biased sampling. Especially, we

consider the learning problem in (i) variable selection, and (ii) classification settings, and will conclude that the conventional approach for learning lacks efficiency and describe how this can be fixed.

2 Training data

Throughout this note, uppercase letters denote one-dimensional random variables, while bold uppercase denotes random vectors. For any subject i, we define the following: The response variable, i.e., time-to-event, represented by U_i. The length-biased variables will be marked with a tilde, e.g., $\tilde{U}_i$ refers to the length-biased survival time. Naturally, we assume that $U_i, \tilde{U}_i \geq 0$. Also, $\mathbf{Z}_i = (Z_{i_1}, Z_{i_2}, \ldots, Z_{i_d})$, with $d \geq 1$, is a vector of covariates. To distinguish biased covariates, we shall use the superscript "$*$". Realizations of random variables and vectors are shown via lowercase and bold lowercase, respectively. To avoid cumbersome notation, no distinction between unbiased and biased realizations is made. When applicable, regression coefficients are denoted by $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_d)^\top$. Nevertheless, $\boldsymbol{\theta}$ is used to indicate the vector of all parameters to be estimated, including the regression ones. Define T_i as the *current lifetime*, i.e., the time interval between the initiating event and sampling time. Similarly, the *residual lifetime*, denoted by R_i, is defined to be the time interval from the sampling until the terminating event. Therefore, $U_i = T_i + R_i$. Also, let C_i denote the *censoring time*, which is the time elapsed from the sampling of the subject until its possible censoring. One may observe only $R_i \wedge C_i = \min(R_i, C_i)$ due to possible censoring. Additionally, we define $X_i := T_i + (R_i \wedge C_i)$. A failure indicator δ_i is defined to be a Bernoulli random variable indicating whether a subject has failed or censored; that is, $\delta_i = \mathbb{1}_{\{R \leq C\}}$. Finally, the training data is assumed to be of the following form:

$$\tilde{S}_c = \{(\tilde{T}_i, \tilde{R}_i \wedge C_i, \delta_i, \mathbf{Z}_i^*) : i = 1, \ldots, n\},$$

with $\mathbf{Z}_i^* = (Z_{i_1}^*, \ldots, Z_{i_d}^*)$. Recall that $\tilde{S}_c$ is length biased (see Figure 1).

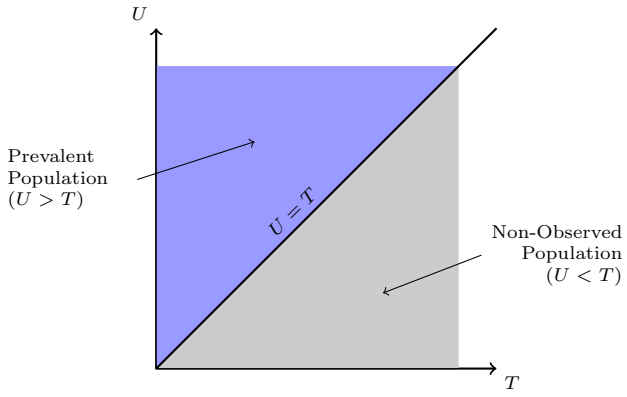


Figure 1: Prevalent vs. Incident Populations: The upper triangle depicts the prevalent population, while the incident population consists of both upper and lower triangles

3 Conditional vs. joint approaches

Consider the regression problem

$$U_i = f_{\boldsymbol{\beta}}(\mathbf{Z}_i) + \boldsymbol{\varepsilon}_i, \; i = 1, \ldots, n \tag{1}$$

where U_i is the response, $\mathbf{Z}_i \in \mathbb{R}^d$ a vector of covariates, $\boldsymbol{\beta} \in \mathbb{R}^{d+1}$ vector of coefficients, $f_{\boldsymbol{\beta}}$ a real-valued function of $\mathbf{Z}_i$, and ε_i a suitable error term independent from $\mathbf{Z}_i$. The core of the conventional approach is utilizing the conditional distribution of U_i given $\mathbf{Z}_i$ for estimation. Bergeron et al. [2] showed that, with left truncation, the conditional likelihood $\mathcal{L}_I$ yields biased estimation because it ignores the information carried by the selection-biased covariates. Moreover, they established that, in contrast, grounding the analysis in the joint distribution of the covariates and response incorporates

this information into the analysis and, consequently, produces superior estimation. Let $\mathcal{L}_J$ denote the joint likelihood. Note that $\mathcal{L}_J(\boldsymbol{\theta}) = \mathcal{L}_I(\boldsymbol{\theta})\, p_{\mathbf{Z}^*}(z)$, with $p_{\mathbf{Z}^*}(z)$ representing the distribution of the biased covariate $\mathbf{Z}^*$. Following the same view as Bergeron et al., we study the impact of the choice of the likelihood function on variable selection and classification.

4 Variable selection

Suppose that in equation (1), $f_{\boldsymbol{\beta}}(\mathbf{Z}_i) = \beta_0 + \beta_1 Z_1 + \cdots + \beta_d Z_d$. Although, training data $\tilde{S}_c$ contains d-dimensional features $\mathbf{Z}_i^*$, not all of them are necessarily related to U_i. In other words, if $\boldsymbol{\beta}^0 = (\beta_1^0, \ldots, \beta_d^0)$ is the true underlying regression coefficient, then there may exist some j's with $1 \leq j \leq d$ such that $\beta_j^0 = 0$. Variable selection in this setting involves finding truly non-zero entries of the regression coefficient $\boldsymbol{\beta}$ in (1) using the train data $\tilde{S}_c$. Let an arbitrary model M be denoted by the set of indices of non-zero entries of the corresponding regression coefficient, i.e., for instance, $M = \{j_1, j_2\}$ is the model corresponding to the family of all $\boldsymbol{\beta}$'s with their only β_{j_1}th and β_{j_2}th entries being non-zero, where $1 \leq j_1, j_2 \leq d$. (Note that such $\boldsymbol{\beta}$ is not unique.) Let M^0 denote the true model, i.e., the model corresponding to $\boldsymbol{\beta}^0$, and $\mathcal{M}$ be the set of all candidate models. $\mathcal{M}$ may or may not include M^0. Following [8], we define

- $\mathcal{M}_o := \{M \in \mathcal{M} : M^0 \subseteq M\}$, called the set of *correct* models, and
- $\mathcal{M}_u := \mathcal{M} \setminus \mathcal{M}_o$, set of *incorrect* models.

Correct models might be inefficient because of their superfluous complexity. The optimal candidate model is the least complex model in $\mathcal{M}_o$, denoted by $M^\star$. Selecting a model from $\mathcal{M}_u$ means missing at least one of the true predictors and, hence, must be avoided. Particularly, as far as revealing the true risk factors is the main interest, it is ultimately desirable for a learning algorithm whose purpose is discovering the true non-zero coefficients not to choose an underfitted model, i.e., from $\mathcal{M}_u$. In the literature numerous likelihood-based criteria has been introduced for model selection (which is, in the present setting, equivalent to variable selection). Examples include Akaike Information Criterion (AIC) [1], Bayesian Information Criterion (BIC) [7], etc, among others. It can be shown that, under length-biased and right-censored training data, employing $\mathcal{L}_J$ as the baseline likelihood function for variable selection is more efficient compared to its counterpart $\mathcal{L}_I$. Roughly, a selection criterion based on $\mathcal{L}_J$ takes less samples to find $M^\star$ in comparison to it being based on $\mathcal{L}_I$. In addition to theoretical justifications, simulation studies, presented in section Simulation Study, also supported this suggestion.

5 Classification

Recently, adopting machine learning techniques has attracted huge attention amongst researchers and policy makers in health care related domains. To a considerable extent, this has been attributed to the swiftly increasing availability of patient records through electronic health records data (EHR). EHR provide access to a large amount of rich data extracted from clinical and administrative data bases. An important question in patient care management is to predict the risk of experiencing a certain outcome, e.g., recurring a health condition, within a particular time frame, say 1 year. However, due to the specific properties of EHR, including length bias and censoring, most of well-known learning techniques cannot be applied naively. Several ad hoc approaches have been tried previously to adapt some machine learning techniques to EHR data but these methods either involve even further loss of information, e.g., by ignoring censored objects, or require the data to be tweaked unnaturally (see [10]). On the other hand, there have been several successful treatments of right-censored data using, for instance, support vector machines, decision trees, and random forests (see, e.g., [3, 4, 5, 6, 9], among others). What is mostly missing in the literature is difficulties induced by left-truncation. This is another problem that we try to address appropriately when it comes to the aforementioned classification question. That is, in presence of length bias how one may correctly model the occurrence of the terminating event in a certain time interval, especially, we focus on the selection bias imposed to the covariates. This may play an important role in methods that rely essentially on the characteristics of the covariate or input space such as tree-based methods.

6 Simulation study

Figure 2 demonstrates the results obtained from BIC variable selection, based on $\mathcal{L}_I$ and $\mathcal{L}_J$ (BIC_I, BIC_j), on simulated, length-biased data generated as follows: (1) Fix $\lambda > 0, \boldsymbol{\beta}$, and sample size n; (2) generate $\mathbf{Z} \in \mathcal{Z} := \{0,1\}^2$ according to discrete uniform distribution over $\mathcal{Z}$; (3) generate U such that $U|\mathbf{Z} \sim \text{Exp}(\lambda e^{\mathbf{Z}\boldsymbol{\beta}})$; and (4) truncate and censor the data based on pre-decided criteria.

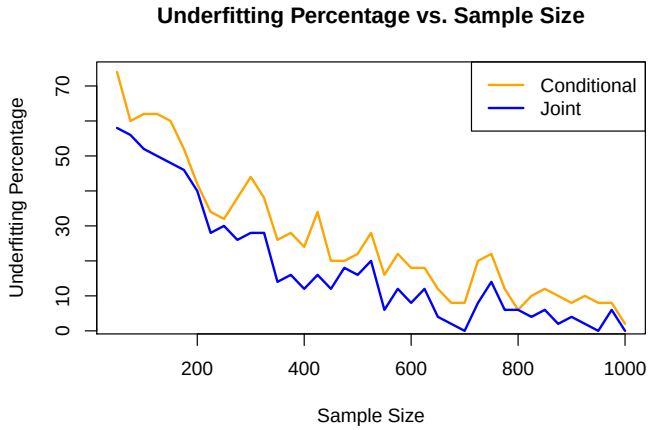


Figure 2: As n increases, underfitting percentage for both BIC_I and BIC_j approaches 0 (consistency)

Size n has been gradually increased from $n = 50$ to 1000 by steps of 25. For each size, 50 different datasets were randomly generated. Figure 2 compares the estimated probability of selecting an underfitted model, i.e., a model containing either Z_1 or Z_2, applying BIC_I and BIC_J. As depicted, BIC_J constantly exhibits a slightly superior performance.

References

[1] H. Akaike. A new look at the statistical model identification. IEEE Transactions on Automatic Control, 19(6):716–723, 1974.

[2] Pierre-Jérôme Bergeron, Masoud Asgharian, and David B Wolfson. Covariate bias induced by length-biased sampling of failure times. Journal of the American Statistical Association, 103(482):737–742, 2008.

[3] Yair Goldberg and Michael R. Kosorok. Support vector regression for right censored data. Electron. J. Statist., 11(1):532–569, 2017.

[4] Hemant Ishwaran, Udaya B. Kogalur, Eugene H. Blackstone, and Michael S. Lauer. Random survival forests. Ann. Appl. Stat., 2(3):841–860, 09 2008.

[5] F. M. Khan and V. B. Zubek. Support vector regression for censored data (svrc): A novel tool for survival analysis. In 2008 Eighth IEEE International Conference on Data Mining, pages 863–868, Dec 2008.

[6] Margaux Luck, Tristan Sylvain, Joseph Paul Cohen, Héloïse Cardinal, Andrea Lodi, and Yoshua Bengio. Learning to rank for censored survival data. CoRR, abs/1806.01984, 2018.

[7] Gideon Schwarz. Estimating the dimension of a model. Ann. Statist., 6(2):461–464, 03 1978.

[8] Jun Shao. Linear model selection by cross-validation. Journal of the American Statistical Association, 88(422):486–494, 1993.

[9] Pannagadatta Shivaswamy, Wei Chu, and Martin Jansche. A support vector approach to censored targets. pages 655–660, 11 2007.

[10] David M. Vock, Julian Wolfson, Sunayan Bandyopadhyay, Gediminas Adomavicius, Paul E. Johnson, Gabriela Vazquez-Benitez, and Patrick J. O'Connor. Adapting machine learning techniques to censored time-to-event health record data: A general-purpose approach using inverse probability of censoring weighting. Journal of Biomedical Informatics, 61:119–131, June 2016.

10 Uncertainty transfer with knowledge distillation

Ibtihel Amara
James J. Clark

Centre for Intelligent Machines, McGill University,
Montréal (Québec), Canada, H3A 2A7

iamara@cim.mcgill.ca
clark@cim.mcgill.ca

April 2020
Les Cahiers du GERAD
G-2020-23-EIW10

Abstract: *Knowledge distillation is a technique that consists in training a student network, usually of a low capacity, to mimic the representation space and the performance of a pre-trained teacher network, often cumbersome, large and very high capacity. Starting from the observation that a student can learn about the teacher's ability in providing predictions, we examine the idea of uncertainty transfer from teacher to student network. We show that through distillation, the distilled network does not only mimic the teacher's performance but somehow captures the original network's uncertainty behavior. We provide experiments validating our hypothesis on the MNIST dataset.*

1 Introduction

The complexity of DNNs demands model compression. It is certain that training these very deep networks can be done with high performance CPUs or even with multiple GPUs. However, these trained models face a challenging situation during deployment wherein they are supposed to fit in terms of memory in small sensors, portable devices and are assumed to provide real-time predictions especially for critical applications such as decision making for autonomous driving. *Model compression* is a method of obtaining smaller and lower memory networks, trained to mimic the behaviour of the original trained large networks. There are different ways to achieve model compression in the literature. Some approaches concentrated on network pruning such as Weight quantization [9] and Low-rank approximation [19]. Other methods focused on network distillation [3, 10, 13, 16]. The latter is a method involving a teacher-student training. The student network, usually a shallow one with few parameters, is trained to replicate the performance of the teacher network, a very deep network. Over the past years, there have been many uncertainty estimation methods for Deep Learning models proposed. Since these deep models are being used for complicated tasks and risk-related applications, estimating uncertainty in these models has become as important as achieving high performance and accuracy. There are different types of uncertainty in Deep Learning [11] and there are various ways of extracting these from models [6, 18]. What all these trends have in common is that they focus on extracting uncertainty using the original network, which can often be very slow. However, in applications where execution speed and memory are crucial such in mobile devices, there should be an alternate way to acquire uncertainty without going through the original large network. In this work, we investigate whether distillation training results into uncertainty transfer from the teacher to the student network. The idea is that a student trained through the distillation process could be able to estimate the teacher network's capacity in providing prediction. In other words, the student can measure the teacher's uncertainty level given an input sample.

2 Related work

2.1 Knowledge Distillation (KD)

The distillation process involves teacher-student training. The student network usually consists of a smaller and shallower network (i.e. fewer parameters). It is trained to replicate the performance of the teacher network, which itself consists of a large and deep network. There have been different variants where different distillation losses were proposed. The work in [2] shows that it is possible to compress the information from an ensemble of networks into a single one. Some research looked at the architecture of the fully connected deep networks and distilled their behavior onto shallow but wider hidden units [1]. Hinton et al. [10] proposed a network that involves the use of parameter called the *temperature factor* at the level of logits. This temperature parameter determines how much the activations of incorrect classes are encouraged. There is also the work of Chen et al. [4] that involved the learning of a "function-preserving" transformation to initialize the parameters of a larger student network. Their goal was to achieve a faster training of deep neural networks.

2.2 Uncertainties in deep learning

There are different types of uncertainty in Deep Learning [11]: aleatoric, which captures the noise related to the observations, and epistemic, which captures uncertainty about the model parameters. Aleatoric uncertainty is the ambiguity about the observation caused by a noisy dataset. An example of aleatoric uncertainty in images would be due to occlusions. Epistemic uncertainty is associated with our incapacity to explain which model parameters are responsible for the set of model outputs. This uncertainty can be explained away if enough data is given to the model [6]. Epistemic uncertainty is important for critical applications and for models that are trained on very small datasets. There have been many methods to quantify uncertainty in Deep Learning. We can categorize these uncertainty measures into three main groups: True Bayesian framework, Bayesian Approximation framework, and the non-Bayesian framework. The Bayesian formalism naturally incorporates uncertainty modelling with probability distributions, which shows the degree of belief regarding of the unknown parameters. Exact Bayesian methods are known to be computationally expensive and complicated especially when dealing with millions of parameters [7]. Therefore, a variety of methods have been developed to overcome this problem, such as variational approximations. The most used techniques under the Bayesian approximation framework are the Monte-Carlo dropout [6] and Monte-Carlo batch normalization[18]. For the non-Bayesian framework for quantifying uncertainty, methods based on ensembles such as Deep ensembles [12] can be used.

3 Methods

3.1 Distillation

In this paper we use the method of knowledge distillation proposed by [10]. To train the distilled network, we are in need of both of hard targets, which are the true labels of the dataset, and the soft targets, which are the class probability produced by the teacher network. This method shows that the logits, which are the inputs to the final softmax, can be used for training small models. These soft targets are obtained through the softmax function that uses a temperature value. A higher value of this temperature parameter produces softer probabilities over classes. The distillation process is then performed through the minimization of the average of two objective functions: (1) cross entropy with the soft targets with high temperature for the softmax and (2) cross entropy with the correct labels using the logits in the softmax.

3.2 Model uncertainty with Monte-Carlo dropout

We used the method of MC dropout to capture model uncertainty. This method, in practice, consists of having a deep network trained using dropout. At test time, we use dropout to sample N networks and record their predictions. The variance of these outputs is the model uncertainty. Please refer to the original work [6] for more details on the mathematical formalism.

4 Experiments and results

4.1 Experimental set up

To investigate our hypothesis that distilled networks can estimate the model uncertainty of the original deep network, we conducted experiments where we considered a VGG16-like network, trained for classification with dropout (having a fixed probability of $p = 0.5$, since this is considered to be the optimal range [17]) using the MNIST dataset. We performed distillation training onto three different student networks in which we varied the number of learnable parameters. The teacher network has 16 layers with $14,913,226$ learnable parameters. The smallest student network (SN1) has four layers with $4,265,066$ parameters. The student network (SN2) has seven layers with $4,721,610$ parameters. Finally, student network 3 (SN3) has the exact same architecture as the teacher network (i.e. 16 layers

and 14, 913, 226 parameters). To evaluate the captured model uncertainty of these different student networks we use these following measures.

Calibration Curves To see if a model is well calibrated, we often use *calibration curves*, sometimes called *reliability diagrams* [5, 15]. These diagrams are a way to represent model calibration. They represent expected accuracy as a function of the confidence (i.e. softmax probabilities). A "perfectly" calibrated model is represented by the identity function. A deviation above (under-confidence) or below (over-confident) the diagonal shows a miscalibrated model. To plot this curve, the confidence level (i.e. probability predictions of the network) are regrouped into M bins of size 10.

Expected Calibration Error In contrast to the calibration curves, which are visual diagrams capturing the degree of model calibration, the Expected Calibration Error (ECE) [14] is a loss function that returns a scalar value that can capture the degree of miscalibration of a particular deep learning model. Similarly to the calibration curves, confidence values are partitioned into $M = 10$ equal bins and we take the weighted average of the difference between the accuracy and confidence at each bin. Full details about this metric can be found at [8].

Mean Uncertainty Prediction Error This measures the mean squared error between the predicted confidence (uncertainty) of the teacher and student networks over the test dataset.

4.2 Results and discussion

We provide in Table 1 the classification accuracy for each of the teacher and student networks trained with KD (KD-SN1, KD-SN2, and KD-SN3) and without KD for student network 2 (SN2). We can observe that each of the distilled student network was able to closely represent and mimic the teacher's performance on all sets of the MNIST dataset. In fact, as these student models get deeper and larger, we can note a decrease in the accuracy. This decrease within the different student networks could be explained by the fact that the MNIST dataset does not require very deep models to achieve high performance. Instead, a smaller network can perform better. Through the use of soft targets, we would expect the larger student network (KD-SN3) to have at least the closest performance to the teacher network since they share the same architecture and capacity. However, we see that distilling a large network onto a large student network does not always lead to similar performance. The student networks SN2 trained without KD and KD-SN2 trained with KD have similar performance on the dataset.

Table 1: Summary table of Classification Accuracies for all networks, Expected Calibration Error (ECE), and Mean Uncertainty Prediction Error (MUPE)

	TN	KD-SN1	KD-SN2	SN2	KD-SN3
Training acc.	99.87%	99.75%	99.52%	99.99%	98.75%
Validation acc.	99.08%	98.93%	98.80%	99.44%	98.08%
Test acc.	99.21%	99.14%	99.04%	99.51%	98.36%
ECE	0.0027	0.0064	0.0019	0.0036	0.0029
MUPE	—	0.0037	0.0019	–	0.0016

One question that can be asked here is: *If we could train a student network to perform as accurate as the teacher model (in giving similar class accuracy) can this student exhibit similar uncertainty behavior as the teacher network?*

To attempt to provide an answer to this question, we performed 100 Monte Carlo (MC) dropout at test time on each of the teacher and student networks then record their predictive mean confidences and plot their corresponding calibration curves. Figures 1, 2, and 3 show, respectively, the calibration graphs MC KD-SN1, MC KD-SN2, and MC KD-SN3 overlayed on the MC TN calibration curve. We chose to plot the MC networks to better apprehend the model uncertainty behavior that underlies within each of these networks.

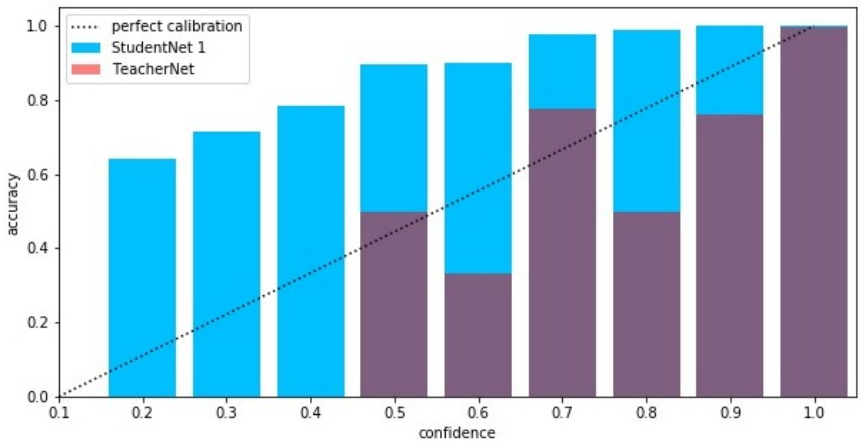


Figure 1: Calibration curve of the MC Student Network 1 (light blue) overlaid on the calibration of MC Teacher Network (red)

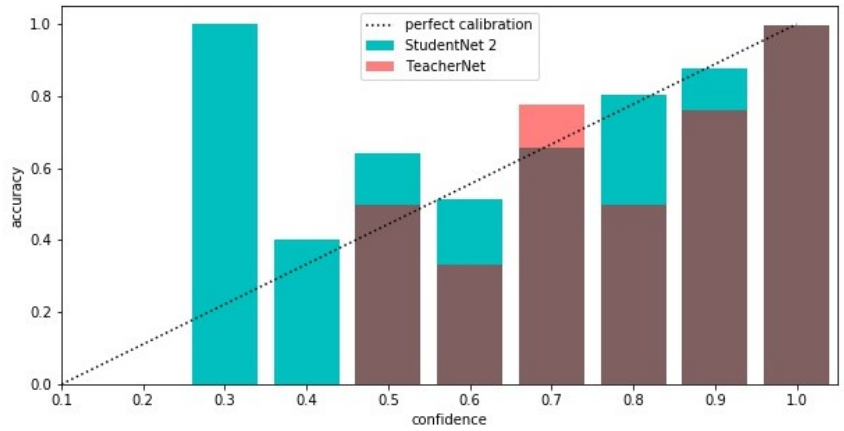


Figure 2: Calibration curve of the MC Student Network 2 (blue) overlaid on the calibration of MC Teacher Network (red)

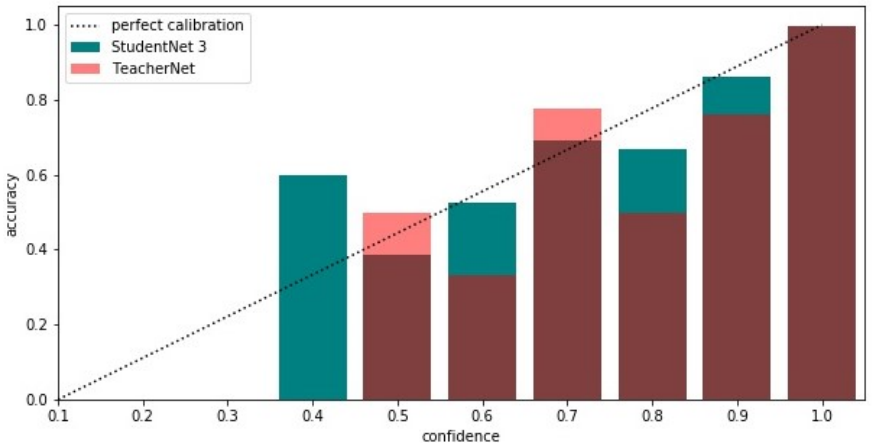


Figure 3: Calibration curve of the MC Student Network 3 (teal) overlaid on the MC Teacher Network (red)

By comparing these graphs, we could observe that as the student network gets deeper and asymptotically closer to the number of parameters of the original network, the calibration curves tend to be as similar as the teacher network. We observe from the figures that the calibration curve of student SN3 is visually closer to the teacher network's calibration graph. This observation can also be analyzed

using the statistical metric ECE. In Table 1, we provide the ECE for each of the teacher and student networks. We see that both student networks KD-SN2 and KD-SN3 are closer to the miscalibration value of the teacher. Specifically, the ECE of KD-SN3 is the closest to the ECE of the teacher network. It is also noteworthy to mention that the ECE of KD-SN2 can also be considered as a close value to the ECE of the teacher model even though this has got better model calibration. Furthermore, the mean uncertainty prediction error can be observed in Table 1 for each of the MC student networks. We note that student KD-SN3 has the lowest mean uncertainty error, which shows that this network reflects the best on the uncertainty behavior of the teacher network. Although both student networks KD-SN2 and KD-SN3 did not provide the closest class accuracy, these distilled networks somehow reflect the teacher's uncertainty. At this stage, we could say that there is a partial uncertainty transfer from teacher to student through the distillation process. However, there are some factors associated with this uncertainty transfer which are mainly the distilled network architecture configuration and the network capacity. Moreover, we see in Table 1 that the ECE for KD-SN2 is lower than the ECE of SN2. This shows that the student network trained with KD is better calibrated. As we have seen, a large student network can almost capture the teacher's uncertainty. Nevertheless, a smaller and compressed network can be used as a higher level insight about the teacher model's uncertainty behavior. Additionally, KD training can be useful for model calibration.

5 Conclusion

We investigated in this paper the possible uncertainty transfer between teacher and student network during distillation training. Our key finding is that uncertainty transfer through distillation can not happen when the student network's capacity is too low.

References

[1] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In Advances in neural information processing systems, pages 2654–2662, 2014.

[2] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 535–541. ACM, 2006.

[3] Zhengping Che, Sanjay Purushotham, Robinder Khemani, and Yan Liu. Distilling knowledge from deep networks with applications to healthcare domain. arXiv preprint arXiv:1512.03542, 2015.

[4] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In International Conference on Machine Learning, pages 2285–2294, 2015.

[5] Morris H DeGroot and Stephen E Fienberg. The comparison and evaluation of forecasters. Journal of the Royal Statistical Society: Series D (The Statistician), 32(1–2):12–22, 1983.

[6] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In international conference on machine learning, pages 1050–1059, 2016.

[7] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 1183–1192. JMLR. org, 2017.

[8] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 1321–1330. JMLR. org, 2017.

[9] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149, 2015.

[10] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015.

[11] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In Advances in neural information processing systems, pages 5574–5584, 2017.

[12] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In Advances in Neural Information Processing Systems, pages 6402–6413, 2017.

[13] David Lopez-Paz, Léon Bottou, Bernhard Schölkopf, and Vladimir Vapnik. Unifying distillation and privileged information. arXiv preprint arXiv:1511.03643, 2015.

[14] Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.

[15] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In Proceedings of the 22nd international conference on Machine learning, pages 625–632. ACM, 2005.

[16] George Papamakarios and Iain Murray. Distilling intractable generative models. In Probabilistic Integration Workshop at Neural Information Processing Systems, 2015.

[17] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1):1929–1958, 2014.

[18] Mattias Teye, Hossein Azizpour, and Kevin Smith. Bayesian uncertainty estimation for batch normalized deep networks. arXiv preprint arXiv:1802.06455, 2018.

[19] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 7370–7379, 2017.

11 State of compact architecture search for deep neural networks

Mohammad Shafiee [a,b]
Andrew Hryniowski [a,b]
Francis Li, [b]
Zhong Qiu Lin [a,b]
Alexander Wong [a,b]

[a] *Waterloo Artificial Intelligence Institute, Waterloo (Ontario), Canada, N2L 3G1*

[b] *DarwinAI Corp., Waterloo (Ontario) Canada, N2V 1K4*

April 2020
Les Cahiers du GERAD
G-2020-23-EIW11

Abstract: *The design of compact deep neural networks is a crucial task to enable widespread adoption of deep neural networks in the real-world, particularly for edge and mobile scenarios. Due to the time-consuming and challenging nature of manually designing compact deep neural networks, there has been significant recent interest into algorithms that automatically search for compact network architectures. A particularly interesting class of compact architecture search algorithms are those guided by baseline network architectures. In this study, we explore the current state of compact architecture search for deep neural networks through both theoretical and empirical analysis of four different state-of-the-art compact architecture search algorithms: i) group lasso regularization, ii) variational dropout, iii) MorphNet, and iv) Generative Synthesis. We examine these methods in detail based on a number of different factors such as efficiency, effectiveness, and scalability across three well-known benchmark datasets, as well as explore practical considerations.*

1 Introduction

Designing compact deep neural network architectures for edge scenarios is very time-consuming and human insight about optimized macro- and micro-architectures is limited in terms of design granularity. To address this need, a number of algorithmic approaches have been proposed in literature for designing compact neural network architecture. For example, a combination of pruning, quantization and coding techniques have been leveraged to address the storage requirement of a deep neural network. Low-rank matrix factorization is another technique which was applied to approximate the filter structures and convolutional kernels in a deep neural network. Structural learning approach during the training of a model is another trick to learn the filter shape and depth during the training process. Compact architectural search was also addressed via variational Bayesian algorithms. One are of great recent interest is neural architecture search (NAS) [1], leading to a variety of strategies such as evolutionary algorithms, reinforcement learning methods, or Bayesian optimization. For example, Shafiee et al. [6] formulated the compact network architectural search problem via an evolutionary synthesis framework so-called EvoNet. Such methods can be designed to target compact network architectures to great potential effect, which we will refer to as compact architecture search approaches.

In this study, we explore the current state of compact architecture search for deep neural networks by conducting both theoretical and empirical analysis on four different state-of-the-art compact architecture search algorithms across three different datasets based on accuracy, computational complexity, and architectural complexity. This provides us with a detailed perspective of where the field stands in this area of research.

2 Theoretical analysis

A key criteria when selecting the set of algorithms to study for a better understanding of the current state of compact architecture search is scalability. Here, we focus on exploring state-of-the-art compact architecture search algorithms that, guided by baseline network architectures, produce compact neural network architectures with smaller yet structured topologies that tend well to parallel computing on modern processor architectures as well as provide lower effective memory footprint during inference. Furthermore, such approaches have been demonstrated to scale well in producing compact deep neural networks that possess strong modelling accuracy for more complex problems and data.

2.1 Group Lasso regularization

Group Lasso can be used to learn structured sparsity in an efficient way. For compact architecture search purposes, it was applied [7] to regularize network structures such as filters or channels during

training via the following loss function:

$$\mathcal{L}(W) = E_D(W) + \lambda_g \sum_{l=1}^{L} R_g(W^{(l)}) \tag{1}$$

where W represents the set of weights in the network and $E_D(\cdot)$ is the loss of the network on the data D. $R_g(\cdot)$ formulates the group lasso for each layer which enforces the structure sparsity during the training and λ_g is the regularization factor.

2.2 Variational dropout

The variational dropout technique adds a Gaussian stochastic factor on each activation during the training step and the mean and standard deviation of this distribution are learned during the training. The values of these parameters reach to zero for those weights that have no impact on the network output which promotes the sparsity. Therefore, the network weights at the end of training are sparse. However the produced sparsity is unstructured and cannot provide any speed up inference. Here we extend upon this approach and drop those filters (i.e., a set of weights in a layer which produces one activation channel at the output) which are more sparse than a pre-defined threshold to generate the final network architecture. Setting up proper parameters and initialization of the parameters is very important to produce the best results.

2.3 MorphNet

MorphNet [3] performs an iterative shrink-and-expand approach to automatically design the structure of a deep neural network. It utilizes weight regularization on network activations to sparsify the network in the shrinking step, followed by a uniform multiplicative approach on all layers in the expand step. More specifically, a penalty term is applied on the loss function during the training in the shrinking step as follows:

$$\begin{aligned} \mathcal{L}(W) =& E_D(W) + \lambda \mathcal{F}(O_{1:M}) \;\; s.t. \\ & \mathcal{F}(O_{1:M}) = \sum_{L=1}^{M+1} \mathcal{F}(\text{layer } L). \end{aligned} \tag{2}$$

The $\mathcal{F}(\cdot)$ is a regularizer on neurons which can induce some of the neurons to be zeroed out and can be formulated as:

$$\mathcal{F}(\text{layer } L) = C \sum_{i=0}^{I_L - 1} A_{L,j} \sum_{j=0}^{O_L - 1} B_{L,j} \tag{3}$$

where $A_{L,j}$ and $B_{L,j}$ are indicator functions that encode whether the input i or output j are alive or zeroed out. L_1 norm on variables of batch normalization is utilized to provide tractable learning via gradient descent. The batch-norm is applied to each layer which means that each neuron has a particular variable in the batch-norm; setting this variable to zero effectively disables the neuron. The MorphNet method needs hyper-parameter tuning; Tuning the parameters in this method plays a vital rule to produce reasonable results.

2.4 Generative synthesis

The overall goal of the Generative Synthesis method [8] as a compact architecture search method is to learn a generator $\mathcal{G}$ that can synthesize deep neural networks $\{N_s | s \in S\}$, given a seed set S, maximize a universal performance function $\mathcal{U}$ while satisfying quantitative human-specified design constraints and performance targets, as defined by an indicator function $\mathbb{1}_r(\cdot)$. This learning of generative machines

for synthesizing deep neural networks can be formulated as the following constrained optimization problem:

$$\mathcal{G} = \max_{\mathcal{G}} \mathcal{U}\Big(\mathcal{G}(s)\Big) \;\; s.t. \tag{4}$$
$$\mathbb{1}_r\Big(\mathcal{G}(s)\Big) = 1, \forall s \in S.$$

Given the intractability of solving this problem, an approximate solution $\mathcal{G}$ to the constrained optimization problem posed is achieved by leveraging the interplay between a generator-inquisitor pair that work together in a synergistic manner to obtain not only improved insights about deep neural networks (via an inquisitor) as well as learn to synthesize compact deep neural networks (via a generator) in a cyclical and iterative manner, taking into consideration human-specified design and operational requirements (size, accuracy tolerance, performance targets, hardware-level requirements such as channel multiplicity and data precision, etc.) when synthesizing progressively more compact deep neural networks until performance targets and operational requirements are satisfied.

3 Empirical analysis

In this study, we further empirically examine the current state of compact architecture search algorithms for producing deep neural networks. The trade-offs between size, speed, and accuracy are measured by conducting several empirical experiments using well-known benchmark datasets. Based on the experimental results, we study the different performance characteristics of the compact deep neural networks produced by the tested compact architecture search algorithms to gain better insights into the effectiveness as well as the search behaviours of the tested algorithms.

3.1 Datasets & networks

The four state-of-the-art compact network architecture search methods studied here are evaluated using CIFAR-10, CIFAR-100 [4], and ImageNet 64x64 [2], a downsampled variant of ImageNet with over 1.28M training images across 1000 classes. In this study, a 32 layer ResNet architecture is used as the baseline architecture for CIFAR-10 while a deeper 50 layer ResNet architecture is used for CIFAR-100 given that they are more difficult tasks. For ImageNet 64x64, two baseline architectures were used for evaluation: i) ResNet-50, and ii) InceptionV3. For the CIFAR-10, CIFAR-100 and ImageNet 64×64 evaluations, the FLOPs target for all four of the tested methods is set to one-third of the respective baseline network architectures to quantitatively investigate the efficacy of the four methods at producing highly compact deep neural networks with low computational costs.

3.2 Results & discussion

The experimental results for CIFAR-10 and CIFAR-100 are shown in Table 1, while the results for ImageNet 64x64 are shown in Table 2.

CIFAR-10/CIFAR-100: As seen in Table 1, the compact deep neural networks produced by each of the tested compact architecture search algorithms have drastically different neural network architectures with very different performance tradeoffs to meet the ∼48 and ∼74 MFLOPs (1/3 of the baseline) performance target for CIFAR-10 and CIFAR-100.

Conducting a close examination of the performance characteristics of the compact deep neural networks produced using the four tested compact architecture search algorithms allowed for the observation of several interesting insights:

- While all produced networks have similar FLOPs as a result of the FLOPs target, several of the produced neural networks have significant more parameters than others. For CIFAR-10, the deep neural network produced via the Group Lasso approach had noticeably more parameters

than the other three methods (e.g., ∼33% higher than Variational Dropout). For CIFAR-100, the deep neural networks produced by MorphNet and Variational Dropout have more than twice the number of parameters as that produced using Group Lasso and Generative Synthesis.

- While all tested methods have the same FLOPs performance target (1/3 FLOPs of baseline), none of the methods produce networks that hit the target exactly. The deep neural network produced by Generative Synthesis had the lowest number of parameters (by around 3% and 7% lower than others for CIFAR-10 and CIFAR-100, respectively).
- Of the methods, Variational Dropout produced the network with the lowest modeling accuracy. MorphNet and Generative Synthesis produced networks with better modeling accuracy than the Group Lasso and Variational Dropout methods, with Generative Synthesis producing the network with the highest accuracy amongst the tested methods for the given FLOPs target. In fact, the network produced by Generative Synthesis achieved the same accuracy as the baseline architecture in the CIFAR-10 case. which illustrates that it was able to find the most balanced trade-off between size, speed, and accuracy.

Table 1: Results for CIFAR-10 and CIFAR-100 across four tested algorithms. Best results in bold

	CIFAR-10			CIFAR-100		
Methods	**MFLOPs**	**#Parameters**	**Top-1(%)**	**MFLOPs**	**#Parameters**	**Top-1(%)**
Baseline	144.369	487,754	91.6	223.946	769,476	67.8
Group Lasso [7]	46.282	225,564	86.7	74.852	268,609	61.6
MorphNet [3]	46.280	153,179	88.8	74.995	435,396	63.3
Variational Dropout [5]	46.046	**151,238**	84.9	74.410	563,197	57.9
Generative Synthesis [8]	**44.658**	152,668	**91.6**	**70.800**	**214,692**	**64.8**

Table 2: Results for ImageNet 64x64 using two baseline architectures. Best results in bold

Architectures	ResNet-50			InceptionV3		
Methods	**MFLOPs**	**#Parameters**	**Top-1(%)**	**MFLOPs**	**#Parameters**	**Top-1(%)**
Baseline	2,629.21	25,593,400	60.0	7,515.7	22,895,000	61.8
Group Lasso [7]	896.71	9,079,334	50.6	2506.7	8,160,687	57.6
MorphNet [3]	824.91	9,294,711	52.9	2490.6	9,489,201	58.5
Variational Dropout [5]	873.29	8,850,781	43.3	2,508.0	8,160,687	53.9
Generative Synthesis [8]	**802.14**	**3,293,420**	**57.8**	**2,340.3**	**5,257,960**	**62.3**

ImageNet 64x64: As seen in Table 2, similar to the CIFAR-10/CIFAR-100 experiments, the compact deep neural networks produced by each of the tested compact architecture search algorithms produce drastically different neural network architectures with very different performance tradeoffs to meet the ∼876 MFLOPs and ∼2505 MFLOPs (1/3 of the baseline) performance targets for ResNet-50 and InceptionV3, respectively.

Experimental results show several interesting insights regarding the performance characteristics of the compact deep neural networks produced using the four compact architecture search algorithms:

- The MorphNet approach produced the largest networks for both cases (in terms of number of parameters). For example, MorphNet produced a network architecture that has ∼2.8× the number of parameters as Generative Synthesis for the ResNet-50 case.
- While all methods have the same FLOPs performance target (1/3 FLOPs of baseline), results demonstrate that, as with the previous experiment, none of the methods produce networks that hit the target exactly. However, Generative Synthesis was able to achieve lower than expected FLOPs (by ∼7% lower for both cases) in this experiment as well.
- As with CIFAR-10 and CIFAR-100, MorphNet and Generative Synthesis produced networks with better modeling accuracy than the Group Lasso and Variational Dropout methods. In both cases, Generative Synthesis produced neural networks that had the highest accuracy and lowest number of parameters amongst the tested methods for the given FLOPs target (e.g., ∼4.9%

higher accuracy than MorphNet in the ResNet-50 case) on ImageNet 64x64. In fact, the neural network produced by Generative Synthesis outperformed the baseline architecture in terms of accuracy by 0.5% in the InceptionV3 case. This illustrates that it was able to find the most balanced trade-off between size, speed, and accuracy.

4 Practical considerations

In order to take advantage of the aforementioned compact architecture search algorithms, there are a number of practical challenges unique to each method that must be considered to achieve good performance in the resulting deep neural networks. One of the biggest challenges to effectively leveraging the Group Lasso regularization technique as a compact architecture search algorithm is to identify the optimal set of hyperparameters, with one of the most crucial hyperparameters being a proper and optimal regularization factor λ_g during the training process. Much like Group Lasso, Variational Dropout also suffers from the *curse of hyper-parameterization*. Selecting the correct regularization strength is key for allowing the model to learn. Too high a rate and a model will not learn at all, too low a rate and the model is not fully learning what parameters should be dropped. MorphNet follows an iterative process of sparsifying a network and linearly growing the network back up. The hyper-parameters that control each step in a given iteration can differ from previous iteration. Selecting the correct rate to sparsify a network is performed via trial and error. Determining when to stop sparsifying a network during any given iteration can be unclear. Generative Synthesis employs an iterative process to learn to generate compact deep neural networks that meet operational requirements and desired performance targets. As such, while Generative Synthesis will iterate until it hits the desired performance targets, a practical challenge with Generative Synthesis is that the number of iterations can vary depending on the complexity of the neural network architecture, the desired performance targets, as well as the complexity of the underlying data.

References

[1] Bowen Baker and et al. Designing neural network architectures using reinforcement learning. 2016.

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. IEEE, 2009.

[3] Ariel Gordon and et al. Morphnet: Fast & simple resource-constrained structure learning of deep networks. 2018.

[4] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[5] Dmitry Molchanov and et al. Variational dropout sparsifies deep neural networks. 2017.

[6] Mohammad Javad Shafiee and et al. Deep learning with darwin: evolutionary synthesis of deep neural networks. 2018.

[7] Wei Wen and et al. Learning structured sparsity in deep neural networks. 2016.

[8] Alexander Wong and et al. Ferminets: Learning generative machines to generate efficient neural networks via generative synthesis. 2018.

12 Deep learning for proactive cooperative malware detection system

Adel Abusitta [a]
Omar Abdel Wahab [b]
Talal Halabi [c]

[a] *McGill Executive Institute, McGill University, Montréal (Québec) Canada, H3A 1G5*

[b] *Université du Québec en Outaouais, Gatineau (Québec) Canada, J8X 3X7*

[c] *Department of Applied Computer Science, University of Winnipeg, Winnipeg (Manitoba), Canada, R3B 2E9*

adel.abusitta@mcgill.ca
omar.abdulwahab@uqo.ca
t.halabi@uwinnipeg.ca

April 2020
Les Cahiers du GERAD
G-2020-23-EIW12

Abstract: *The past few years have seen the ability of cooperative Malware Detection Systems (MDS) to detect complex and unknown malware. In a cooperative setting, an MDS can consult other MDSs about suspicious malware and make a final decision using an aggregation mechanism. However, large delays may arise from both applying an aggregation mechanism and waiting to receive feedback from consulted MDSs. These shortcomings render the decisions produced by existing cooperative MDS approaches ineffective in real-time. To address the above-mentioned problem, we propose a deep learning-based cooperative MDS that efficiently exploits historical feedback data to foster proactive decision-making. More specifically, the proposed approach is based on Denoising Autoencoder (DA), which allows us to learn how to reconstruct complete MDSs' feedback from partial feedback. Our results show the effectiveness of the proposed framework on a real-life dataset.*

1 Introduction

The current communication and computing infrastructure is becoming more and more complex and vulnerable to cyber attacks. In the recent years, studies and results have shown that the use of cooperative Malware Detection Systems (MDSs) can enhance the detection accuracy compared to traditional single MDSs [1, 4, 5]. This is such since it is becoming increasingly difficult for one single MDS to detect all existing malware [1, 2, 5], due to its limited knowledge of such malware patterns and implications. The cooperation among MDSs can be achieved through allowing them to exchange their malware analysis feedback and exploit each other's expertise to cover unknown malware patterns, thus achieving mutual benefits.

There are considerable delays associated with adopting existing cooperative MDS approaches [1, 5]. These delays are mostly due to the computation complexity of using aggregation algorithms such as Bayesian Theory, and also the large geographic distances that separate the MDSs. In fact, each MDS, after receiving feedback from consulted MDSs regarding a suspicious malware, is required to use a suitable feedback algorithm, in order to make a final decision about the suspicious malware. The aggregation method is usually costly in terms of computation time and depends on many factors such as the number of consulted MDSs, and MDSs' trust levels and expertise [1, 2, 5]. In addition, due to the uneven MDSs' connections and communication speeds and other unknown factors (e.g., busy MDSs, compromised MDSs), there is no guarantee that feedback will be synchronously received. Therefore, decisions on whether or not to raise an alarm regarding some suspicious malware might be excessively delayed due to the missing feedback of a single MDS. Hence, the decisions generated by the cooperative MDS are ineffective in a real-time setting.

To address the above-mentioned limitations, we propose a proactive cooperative MDS that integrates a deep learning approach. The proposed approach exploits the historical MDSs' feedback to predict the status of a certain suspicious malware. This is done proactively without having to apply any aggregation mechanism on consulted MDSs' feedback, nor having to wait until receiving all the feedbacks from the consulted MDSs, i.e., only partial and/or incomplete feedback can be used to predict the status of suspicions attack. This, in turn, makes our approach reliable and feasible in real-time environments, where decisions on malware must be rapidly taken in order to effectively apply the required action measures at the right time. More particularly, the proposed approach is based on Stacked Denoising Autoencoders (SDAE), where a denoising autoencoder is used as a building block to train a deep neunetwork [11, 12]. We capitalize on the fact that a denoising autoencoder can learn how to reconstruct original inputs giving partial data inputs, through allowing deep neural networks to learn how to extract features that are robust to incomplete MDSs' feedback. Our contributions are summarized as follows:

- Proposing a cooperative malware detection system that enables decision-making on suspicious malware, even with partial MDSs's feedback.
- Designing a proactive cooperative MDS, which enables us to make decisions about suspicious malware proactively, i.e., without the need to apply aggregation mechanisms on MDSs' feedback.

2 SDAE-MDS: The proposed approach

In this section, we first present the concept of traditional autoencoders. Then, we explain the proposed approach.

2.1 The traditional autoencoders

An autoencoder is an unsupervised learning method that is used to learn reliable data codings [9]. It is used to pre-train each layer in a deep neural network in order to obtain better initial weights that lead to a better-performing classification [3]. Researchers have reported that weights initialization using autoencoders can improve the performance of deep neural networks, compared to a random initialization [3].

An autoencoder is used as a building block for deep networks [3]. In particular, it takes an input vector (MDSs' feedback) $x \in [0,1]^d$, where d is the vector dimension, and maps it to a hidden representation $h \in [0,1]^{d'}$ using the following equation:

$$h = f_\theta(x) = Sig(W * x + b) \tag{1}$$

$\theta = \{W, b\}$, W is a weight matrix and b is a bias vector. Thereafter, the resulting hidden layer representation h will be reconstructed to the output layer x' using a decoding function as follows:

$$x' = g_{\theta'}(h) = Sig(W' * h + b') \tag{2}$$

$\theta' = \{W', b'\}$, W' and b' are a weight matrix and a bias vector of the reverse mapping, respectively. The purpose of the model is to optimize the parameters of the model, so that the reconstruction error between the input and output can be reduced [6].

2.2 The proposed approach

In order to make an autoencoder robust to incomplete MDSs' feedback, the autoencoder should be trained to reconstruct its MDS's feedback even if the feedback does not represent the whole MDSs' feedback (i.e., when some feedback are not available). The autoencoder that deals with corrupted version of the input is called a denoising autoencoder [11]. This is achieved by adding noise to the initial input x before passing it to the hidden layer. The objective is to reconstruct x, where x represents the MDSs' feedback. Thus, a partially corrupted version z will be obtained from x as follows: $z = alpha(x)$ where $alpha$ is a corruption mechanism [11]. In our model, we use Masking Noise Approach (MNA) for the corruption process, as it is useful to represent incomplete MDS's feedback [12]. In MNA noise, a fraction v (selected at random) of each MDSs' feedback x is forced to be 0, while the others remain untouched. In fact, other noise can also be used such as Gaussian noise. However, MNA noise is more useful to simulate incomplete MDSs' feedback [12] since the noise will only change partial feedback.

The autoencoder is then used to take corrupted data z and attempt to learn how to reconstruct x. This is done by allowing the input z to be mapped to a hidden representation, i.e.,

$$h = f_\theta(z) = Sig(W' * z + b') \tag{3}$$

Note that we select z as input instead of x since a traditional autoencoder was used. The value of h is then used to reconstruct x' as follows:

$$x' = g_{\theta'}(h) = Sig(W * h + b) \tag{4}$$

The denoising autoencoder architecture is described in Figure 1. As given in the traditional autoencoder, the parameters are trained to minimize the average reconstruction error:

$$\begin{aligned}\theta^*, \theta'^* &= arg \;\; minimize_{\theta,\theta'} \; \frac{1}{n}\sum_{j=1}^{n} L(z^{(j)}, x'^{(j)}) \\ &= \frac{1}{n}\sum_{i=1}^{n} L(z^{(j)}, g_{\theta'}(f_\theta(z^{(j)})))\end{aligned} \tag{5}$$

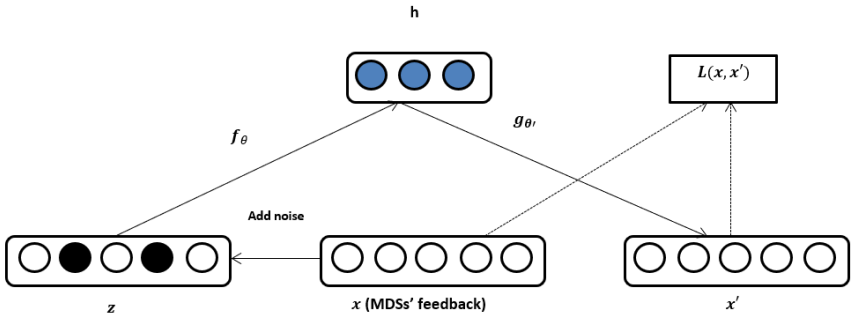


Figure 1: MDS-based denoising autoencoder architecture

The training algorithm of the proposed MDS-based denoising autoencoder is described as follows. For the raw inputs x, we randomly select parts of them to be set to 0 as the corrupted inputs z. The corrupted input z will then be encoded to the hidden code and reconstructed to the output. Note that x' is a deterministic function of z rather than x. The reconstruction that is computed between z and x is denoted as $L(x, x')$. The parameters of the model are randomly initialized and then optimized using stochastic gradient descent algorithms. The above mentioned-steps are performed for each layer added in the proposed MDS-based deep neural network. To generate a classifier for MDS, we add a classifier (e.g., logistic regressions) to the last layer. Then, the parameters of all the layers will be fine-tuned to minimize the error of predicting the target label (i.e., malware or not) using a back-propagation algorithm [3, 7, 8, 11, 12].

3 Evaluation results

To evaluate the proposed model, we create a dataset containing MDSs' feedback on suspicious malware. This dataset was created based on the Android Malware Dataset (MAD) [10], where each 1 or 0 in the new dataset corresponds to the answer of an MDS to a given row of the MAD dataset [10]. The created dataset is used to train the proposed model. Then, the ability of the proposed approach in making decisions about suspicious malware was tested in the presence of partial/incomplete feedback. To represent partial/incomplete MDSs' feedback, some of the MDSs' feedback (selected randomly) were left blank. In this case, blanks indicate that some of MDSs' feedback are yet to be received, due to some unexpected delays (e.g., busy MDS).

The accuracy of the proposed approach is first tested and compared in a complete information scenario, i.e, all MDSs' feedback is received on time. This is useful to evaluate the effectiveness of the proposed approach in making decisions given partial feedback. Figure 2 shows that the accuracy of the proposed model, with a variety of hidden units, was slightly degraded (less than 1.1%). These results suggest that the proposed deep leaning-based approach is able to effectively makes the right decisions on suspicious malware events, even in the presence of some incomplete feedback.

The proposed model (i.e., SDAE-MDS) was also compared with another approach, namely the Stacked Auto Encoder-MDS (SAE-MDS). SAE-MDS uses traditional autoencoders as a building block

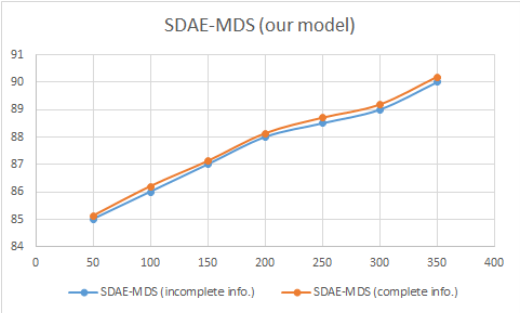


Figure 2: Detection accuracy performance compare to having all the MDSs' feedback (complete information) - number of hidden layers = 3.

for the deep neural networks. The study was conducted with different numbers of layers and hidden nodes. Our model (Figure 3) yields an increased accuracy compared to SAE-MDS (Figure 4). The reason is that we use denoising autoencoders as a building block for our deep neural networks, which allow us to extract robust features that lead to a better classification, despite the incomplete feedback given as inputs to the deep neural network [11]. The denoising autoencoder learned how to reconstruct the feedback from corrupted inputs.

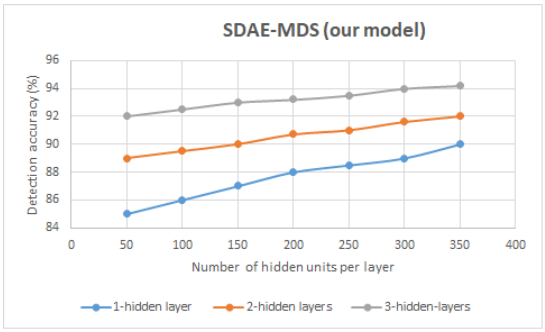


Figure 3: Detection accuracy of SDAE-MDS

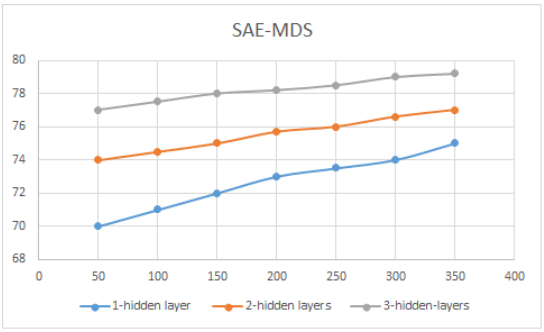


Figure 4: Detection accuracy of SAE-MDS

Figure 5 compares SDAE-MDS (the proposed model) with two other denoising models based on training with noisy input, namely SAE(1)-MDS and SAE(2)-MDS. SAE(1)-MDS is a 3-hidden-layers SAE-MDS where noisy inputs were only used for the pretraining. However, SAE(2)-MDS is also 3-hidden-layers SAE-MDS where noisy inputs were used for both pretraining and fine-tuning. The results demonstrate that our model is also resilient to the increase in the percentage of noises.

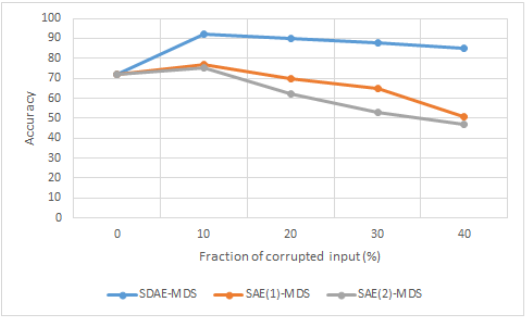


Figure 5: SDAE-MDS vs. training with noisy input

Note that when the corrupted inputs (percentage) equals 0%, all models (SAE(1)-MDS, SAE(2)-MDS and SDAE-MDS) yield the same results in terms of classification accuracy. This is due to the fact that when 0% is applied, the three models will be the same as SAE.

4 Conclusion

We proposed a proactive cooperative MDS. The proposed approach allows us to exploit historical feedback to produce learning models that can effectively and efficiently predict the label (malware or not) of the suspicious malware even when some feedback are missing. The proposed approach is based on stacked denoising autoencoders, where we use a denoising autoencoder as a building block for the deep learning classifier. The proposed MDS-based denoising autoencoder is used to learn how to reconstruct original MDSs' feedback given partial ones. The proposed model can also make decisions regarding suspicious malware without having to apply any aggregation mechanism on the consulted MDSs' feedback. Experimental results show the effectiveness of the proposed approach.

References

[1] Adel Abusitta, Martine Bellaiche, and Michel Dagenais. A trust-based game theoretical model for cooperative intrusion detection in multi-cloud environments. In 2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), pages 1–8. IEEE, 2018.

[2] Adel Abusitta, Martine Bellaiche, and Michel Dagenais. Multi-cloud cooperative intrusion detection system: trust and fairness assurance. Annals of Telecommunications, 74(9–10):637–653, 2019.

[3] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In Advances in neural information processing systems, pages 153–160, 2007.

[4] Áine MacDermott, Qi Shi, and Kashif Kifayat. Collaborative intrusion detection in federated cloud environments. Journal of Computer Sciences and Applications, 3(3A):10–20, 2015.

[5] Carol J Fung and Quanyan Zhu. Facid: A trust-based collaborative decision framework for intrusion detection networks. Ad Hoc Networks, 53:17–31, 2016.

[6] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. Deep learning, volume 1. MIT press Cambridge, 2016.

[7] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. Neural computation, 18(7):1527–1554, 2006.

[8] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. science, 313(5786):504–507, 2006.

[9] Cheng-Yuan Liou, Wei-Chen Cheng, Jiun-Wei Liou, and Daw-Ran Liou. Autoencoder for words. Neurocomputing, 139:84–96, 2014.

[10] Laya Taheri, Andi Fitriah Abdul Kadir, and Arash Habibi Lashkari. Extensible android malware detection and family classification using network-flows and api-calls. In 2019 International Carnahan Conference on Security Technology (ICCST), pages 1–8. IEEE, 2019.

[11] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning, pages 1096–1103. ACM, 2008.

[12] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research, 11(Dec):3371–3408, 2010.

13 Neural network sparsification using Gibbs measures

Alex Labach
Shahrokh Valaee

Department of Electrical & Computer Engineering, University of Toronto, Toronto (Ontario) Canada, M5S 1A1

alex.labach@mail.utoronto.ca
valaee@ece.utoronto.ca

April 2020
Les Cahiers du GERAD
G-2020-23-EIW13

Abstract: *Pruning methods for deep neural networks based on weight magnitude have shown promise in recent research. We propose a new, highly flexible approach to neural network pruning based on Gibbs measures. We apply it with a Hamiltonian that is a function of weight magnitude, using the annealing capabilities of Gibbs measures to smoothly move from regularization to adaptive pruning during an ordinary neural network training schedule. Comparing to several established methods, we find that our network outperforms those that do not use extra training steps and achieves a high accuracy much faster than those that do. We achieve a ¡3% reduction in accuracy on CIFAR-10 with ResNet-32 when pruning 90% of weights.*

1 Introduction

Neural network sparsification via connection pruning has recently been a topic of renewed academic interest. Recent work has shown both the practical utility of pruning in neural network compression systems [6] and theoretical insights gained by experiments with pruning [2]. With modern deep neural network topologies, it is common to be able to prune over 80% of weights with little degradation in performance.

While other criteria have been explored, recent research has focused on pruning weights with low magnitude, since magnitude-based pruning schemes have proven more effective than other established ones [4, 5]. However, existing schemes often require loops of pruning and retraining to achieve good results [6, 2], substantially increasing training time. Our work seeks to approach the performance of such methods without requiring additional training.

Taking inspiration from statistical physics, we propose inducing a Gibbs measure over the weights of a neural network, and sampling from it during training to determine pruning masks. This procedure induces a learned network structure that is resilient to high degrees of pruning. Gibbs measures are highly flexible in terms of network properties that they can express, and quadratic energy functions, such as Ising models, can capture parameter interactions and induce desired structure in pruning masks. They also naturally allow a temperature parameter to be used for annealing, gradually converging to a final pruning mask during training and improving network resilience to pruning. We propose a simple energy function based on weight magnitude to define the Gibbs measure used in our experiments.

2 Proposed method

A Gibbs measure is a probability measure over a vector $\mathbf{x}$ of the form:

$$p(\mathbf{x}) = \frac{1}{Z(\beta)} e^{-\beta H(\mathbf{x})}, \tag{1}$$

where $H(\mathbf{x})$ is the Hamiltonian function, or energy, β is an inverse temperature parameter that can be used for annealing, and $Z(\beta)$ is a partition function that normalizes the measure. In our proposed method, $\mathbf{x}$ represents a pruning mask for a single neural network layer, with $x_i \in \{0, 1\}$. Given the weights of the layer as a flattened vector $\mathbf{w}$, a weight w_i is masked (treated as zero) during training if the corresponding x_i is 0. $\mathbf{x}$ is sampled from $p(\mathbf{x})$ at every training step and once again after training to determine the final mask.

To sample in such a way that weights with lower magnitude are more likely to be masked, we use a Hamiltonian of the form:

$$H(\mathbf{x}) = \sum_i (Q(p, \mathbf{w} \circ \mathbf{w}) - w_i^2) x_i, \tag{2}$$

where $\circ$ represents elementwise multiplication, p is the target pruning fraction, and $Q(p, \mathbf{w} \circ \mathbf{w})$ represents the pth quantile of the values in the vector $\mathbf{w} \circ \mathbf{w}$.

We perform annealing by increasing β from a low value to a high value while training. At high temperatures (low β), differences in the Hamiltonian do not affect sampling much, and so roughly 50% of weights are pruned randomly. This is equivalent to the regularization method dropconnect [10] and starts conditioning the network to be robust under weight pruning. Once annealed to a low temperature (high β), the Gibbs measure converges to pruning the fraction p of weights with lowest magnitude.

This is a highly flexible pruning approach, and although we only consider one Hamiltonian formulation in this paper, others could be designed that take into account characteristics such as network activations or interactions between weights. A Hamiltonian with quadratic terms, such as an Ising model, could induce structure in the pruning masks so as to make them more practically applicable to network compression.

In general, Gibbs measures are computationally expensive to sample from, since the partition function contains many terms. However, since our proposed Hamiltonian is a sum of terms that are each a function of just one element in $\mathbf{x}$, the measure factors into functions of each x_i, and each x_i can therefore be sampled independently. More complex Hamiltonians would likely require similar shortcuts based on independence, or acceleration on GPU or hardware using an efficient sampler such as the Swendsen-Wang algorithm [1] to be realistically usable at each training step.

The goal of our proposed method is to optimize a network's parameters through training at the same time as the final pruning mask over the network is being determined. This means that early on in training, the network learns to be robust under random pruning and receives the regularization benefits of dropconnect, and then gradually converges towards a particular pruning mask, spending a significant amount of later training time adapting to the particular structure of the pruning mask. The balance between time spent on adapting to random masks and time spent adapting to the final mask can be controlled by the particular annealing schedule for β used in training.

3 Experiments

We compare the performance of our proposed method to various baselines and established methods using ResNet-20 and ResNet-32 [7] on the CIFAR-10 [9] dataset. The networks are trained for 200 epochs using the Adam optimizer [8], with a learning rate initially set to 1×10^{-3} and reduced by a factor of 10 every 60 epochs. This achieves baseline top-1 accuracies of 90.7% for ResNet-20 and 91.6% for ResNet-32. We prune all convolutional layers except for the first one, following the recommendation in [4]. When using our proposed method, we anneal β according to a logarithmic schedule from 0.7 to 10000 over the first 128 epochs. These values were chosen empirically to cover a range from an effective pruning rate of 50% to p.

We compare our methods to two kinds of established methods: those that do not add extra training steps to the optimization procedure and those that do. Since our method does not add extra training steps, when comparing to other methods that do not either, we simply compare final accuracies at different p values. This comparison is shown in Tables 1 and 2. One-off pruning simply removes the fraction p of weights in each layer with the lowest magnitude, with no changes to the training procedure. We also test applying a random pruning mask at the beginning of training and maintaining it throughout, effectively training a smaller network from the start. We test using $l1$ regularization with a tuned penalty of 0.001 to induce sparsity during training before masking the fraction p weights in each layer with the lowest magnitude. Finally, we test targeted dropout, which is a recently proposed method described in [5]. We use the most successful hyperparameter settings described in the paper: $\alpha = 0.75, \gamma = 0.9$. The results show our proposed method consistently outperforming the other tested pruning methods at all p values.

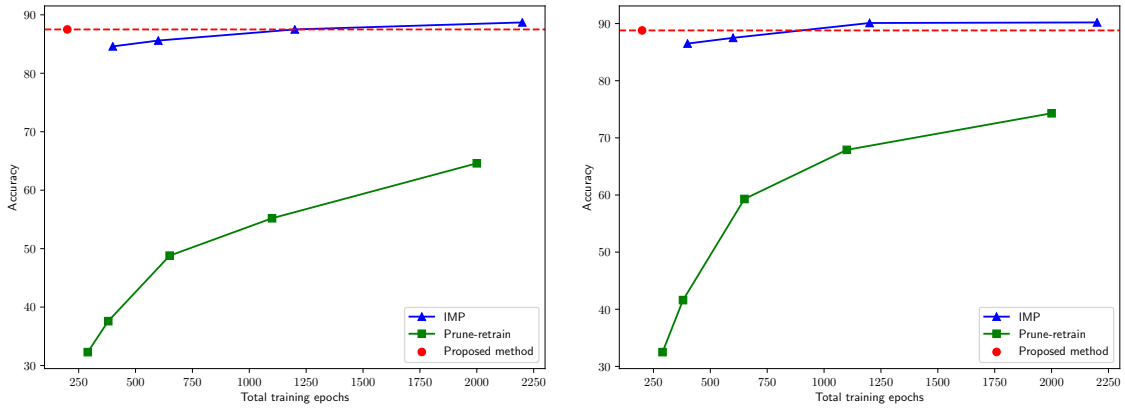
We also compare our proposed method to established pruning methods that require additional training steps beyond the initial network training in Figure 1. In these comparisons, we look at how many training epochs are required for such methods to match the performance of our proposed method.

Table 1: Comparison of our proposed method to other pruning methods that do not use retraining on ResNet-20. Values are top-1 accuracy in percent on CIFAR-10 averaged over two runs

Method	Pruning rate			
	50%	75%	90%	95%
One-off	79.6	17.6	10.0	9.9
Random mask	89.3	87.2	83.2	79.8
l1 loss	81.6	81.7	75.6	43.1
Targeted dropout	82.1	82.1	82.1	21.2
Proposed method	89.9	89.0	87.5	85.2

Table 2: Comparison of our proposed method to other pruning methods that do not use retraining on ResNet-32. Values are top-1 accuracy in percent on CIFAR-10 averaged over two runs

Method	Pruning rate			
	50%	75%	90%	95%
One-off	86.3	29.9	10.0	10.0
Random mask	90.3	88.3	84.8	82.1
l1 loss	81.6	81.1	82.4	55.6
Targeted dropout	87.1	87.1	87.1	26.5
Proposed method	90.5	90.4	88.8	87.1



(a) ResNet-20 (b) ResNet-32

Figure 1: Comparison of our proposed method to other pruning methods that use retraining on ResNet variants. Values are top-1 accuracy in percent on CIFAR-10. All methods prune 90% of weights

The first method we compare to is that proposed in [6], which we call prune-retrain sparsification. In this method, once training is complete, a certain percentage of the weights with lowest magnitude are pruned and the network is retrained for a certain number of epochs. This procedure is then repeated several times, with the pruning percentage increasing each time. We use pruning percentages of 10%, 20%, 30%, etc. up to 90%, retraining for n epochs after each step, leading to a total of $200 + 9n$ epochs. n is varied to test different overall training times. The other method we compare to is iterative magnitude pruning (IMP) [2] with rewinding [3]. This method trains the network several times, pruning gradually more each time and then rewinding the network weights to the values they had after the first 500 training steps. To test different training times, we vary the number of number of times that the network is trained with some pruning rate between the initial rate of 0% and the final rate of 90%.

These results show that established iterative methods require many more training steps to achieve the same results as our proposed method: six times as many on ResNet-20 and over three times as long on ResNet-32 for IMP. Prune-retrain sparsification had far worse performance than our method, even with ten times the number of training epochs. For many applications and datasets, trading a slight reduction in accuracy for being able to avoid expensive retraining schemes could be preferable. It also

might be possible to combine our proposed method with an iterative retraining scheme to produce the same accuracies as IMP with less training time.

In summary, our experiments show our proposed method outperforming several other methods that do not use network retraining, and achieving high performance much more quickly than methods that use network retraining.

4 Conclusion

We introduce a novel neural network sparsification method based on Gibbs measures that achieves high pruning ratios with little reduction in accuracy and without requiring additional training steps. It shows the efficacy of simultaneously training and pruning a network rather than training and pruning as distinct steps at different times. Future work could use this method to accelerate iterative magnitude pruning by converging to an effective pruning mask more quickly. The generality of Gibbs measures also means that similar pruning methods could be developed that induce desired structures in the final pruning mask.

References

[1] A. Barbu and Song-Chun Zhu. Generalizing Swendsen-Wang to sampling arbitrary posterior probabilities. IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(8):1239–1253, 2005.

[2] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In Proceedings of the International Conference on Learning Representations (ICLR), 2019.

[3] Jonathan Frankle, Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis. arXiv preprint arXiv:1903.01611, 2019.

[4] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. arXiv preprint arXiv:1902.09574, 2019.

[5] Aidan N Gomez, Ivan Zhang, Kevin Swersky, Yarin Gal, and Geoffrey E Hinton. Learning sparse networks using targeted dropout. arXiv preprint arXiv:1905.13678, 2019.

[6] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In Proceedings of the International Conference on Learning Representations (ICLR), 2016.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[8] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[9] Alex Krizhevsky. Learning multiple layers of features from tiny images. University of Toronto, 05 2012.

[10] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In International Conference on Machine Learning, pages 1058–1066, 2013.

14 Distributed stochastic gradient descent with quantized compressive sensing

Dipayan Mitra
Ashish Khisti

Department of Electrical & Computer Engineering, University of Toronto, Toronto (Ontario) Canada, M5S 1A1

mitradi2@ece.utoronto.ca
akhisti@ece.utoronto.ca

April 2020
Les Cahiers du GERAD
G-2020-23-EIW14

Abstract: *One of the major challenges in large-scale distributed machine learning involving stochastic gradient methods is the high cost of gradient communication over multiple nodes. Gradient quantization and sparsification have been studied to reduce the communication cost. In this work we bridge the gap between gradient sparsity and quantization. We propose a quantizatized compressive sensing-based approach to address the issue of gradient communication. Our approach compresses the gradients by a random matrix and apply 1-bit quantization to reduce the communication cost. We also provide a theoretical analysis on the convergence of our approach, under the gradient bound assumption.*

1 Introduction

The advent of internet-of-things (IoT) has changed the way data is collected and processed. Millions of connected users are generating huge amount of un-processed data, often sensitive. Distributed processing, exploiting data-parallelism, is often adopted to train a large-scale machine learning model [13, 18, 22]. Chen et al. proposed Synchronous stochastic gradient descent (Sync-SGD), which is often used as a preferred distributed optimization technique [5]. Sync-SGD consists of a centralised *parameter server* and a number of *workers* performing the following tasks:

- Parameter server communicates the model parameters with each worker.
- Each worker, in parallel, computes the gradients on a mini-batch of training data. Gradients are then sent to parameter server.
- Upon receiving gradient updates from all participating workers, parameter server performs a gradient aggregation. Global model parameters are updated based on the aggregated gradients and sent to each worker.

The above process is repeated until the model converges.

Although Sync-SGD performs well while the number of participating workers are scaled up, communication of gradients between the workers and the parameter server causes a bottleneck [10, 21]. In other words, the total time required for one complete iteration of Sync-SGD can be categorized as gradient computation and communication by each worker. Earlier studies have identified gradient communication cost to be more challenging over the gradient computation cost [25, 26, 27]. Hence compressing the gradients to reduce the communication overhead is widely studied.

In literature various techniques involving sparsity and quantization of deep neural networks (DNNs) have been explored [2, 8, 9, 12, 23, 24]. Stich et al. proposed gradient sparsification technique, where each worker sends top-k gradient parameters to the parameter server [15, 19]. Although such heuristic technique works well in practice, scaling up the number of workers leads to poor compression performance and divergence [20]. However, to the best of our knowledge, compressive sensing has not been used, exploiting sparsity of the gradient updates.

In this paper, we propose a quantized compressive sensing approach to reduce the gradient communication time. We use compressive sensing to acquire compressed measurements from the sparse gradient updates. Compressed measurements are further quantized using a 1-bit quantizer, for the ace of hardware implementation. Our work has two major contributions: (1) we propose a quantized compressive sensing-based approach to reduce the the gradient communication cost in the distributed learning, (2) we provide a theoretical analysis on the convergence of the proposed approach.

2 Motivation

Our work is motivated by the observation of sparsity, induced by Rectified Linear Unit (ReLU) activation layers, in DNNs. ReLU activation function takes the following form: $f(x) = \max(0, x)$ and

forces the gradients to be 0 $\forall x < 0$ [11]. As a result on average 44% of the operations performed in most of the modern DNNs, for example AlexNet, GoogLeNet etc., are *ineffective* [17]. In our work, compressive sensing is used to exploit this sparsity of DNNs.

3 Background

3.1 Compressive sensing

Compressive sensing is a sampling technique for signals which are sparse or compressible in some known basis [6]. Let us assume an N dimensional signal $\mathbf{x}$ which is K'-sparse [1], where $K' << N$. The sensing process can be defined as follows,

$$\mathbf{y}_{M\times 1} = \mathbf{\Phi}_{M\times N}\mathbf{x}_{N\times 1} \tag{1}$$

$$= \mathbf{\Phi}_{M\times N}\mathbf{\Psi}_{N\times N}\mathbf{s}_{N\times 1} \tag{2}$$

where $\mathbf{y}$ denotes M dimensional measurement vector, $\mathbf{\Phi}$ and $\mathbf{\Psi}$ denote measurement matrix and the sparsifying basis respectively. Here $\mathbf{s}$ denotes the sparse vector.

Once the compressive measurements are obtained, goal of the reconstruction is to find the sparsest solution from $\mathbf{y}$. Although the sparsest solution can be obtained by solving ℓ_0 optimization problem, it is computationally complex. Instead, in classical compressive sensing ℓ_1 minimization problem is solved to obtain the sparse solution, which is theoretically proven to be equivalent to minimizing ℓ_0 optimization problem [4, 7]. The reconstruction of the compressed measurements can be expressed as follows,

$$\widehat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \|\mathbf{x}\|_1 \quad \text{s.t.} \quad \mathbf{y} = \mathbf{\Phi}\mathbf{x} \tag{3}$$

where $||.||_1$ represents ℓ_1 norm.

3.2 Quantized Compressive Sensing

Boufounos et al. introduced quantized compressive sensing (QCS) where the quantization is modeled as an additive measurement noise, shown in equation 4 [3].

$$\mathbf{y} \quad = \quad Q(\mathbf{\Phi}\mathbf{x}) = \mathbf{\Phi}\mathbf{x} + \mathbf{e} \tag{4}$$

where $Q(.)$ denotes the quantizer. Measurement noise $\mathbf{n}$ is bounded by the quantization interval Δ and the dimension of the compressed measurement (M) as follows [3],

$$||\mathbf{e}||^2 \leq \sqrt{\frac{M\Delta^2}{12}} = \epsilon \tag{5}$$

In QCS reconstructed signal can be obtained by solving,

$$\widehat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \|\mathbf{x}\|_1 \text{ s.t. } \|\mathbf{y} - \Phi\mathbf{x}\|_2 \leq \epsilon \tag{6}$$

A LP-based reconstruction algorithm can be used to obtain $\widehat{\mathbf{x}}$ from the compressed measurements $\mathbf{y}$ [3, 4]. In this work, the reconstruction error β is considered to be a factor accounting both the quantization error (during measurement) and the LP-based reconstruction error. Equation 7 shows the aforementioned noise, which can be bounded by a positive quantity β.

$$||\widehat{\mathbf{x}} - \mathbf{x}||^2 = ||\mathbf{n}||^2 \leq \beta \tag{7}$$

In the next section we discuss the proposed QCS-based gradient compression in distributed learning.

[1] Although in literature sparsity is denoted as K, we use K' to denote sparsity for avoiding conflict with the total number of workers (denoted by K).

4 Proposed approach

4.1 Problem formulation

Let us consider K number of workers participating in a distributed learning process to evaluate parameters $\mathbf{w}$ on training samples $\mathbf{x}$, drawn (i.i.d.) from a probability distribution $dP(\mathbf{x})$. At t-th iteration, a mini-batch of training samples are split and evenly distributed among K workers. Each worker computes its local gradients $\mathbf{g}_t^{(k)}$ with respect to its training samples $\mathbf{x}_t^{(k)}$ and communicates the update with the parameter server to perform aggregation, following: $\mathbf{g}_t = \frac{1}{K}\sum_{k=1}^{K}\mathbf{g}_t^{(k)}$. The aggregated global gradient $\mathbf{g}_t$ is used to evaluate the updated model parameter $\mathbf{w}_{t+1}$ as,

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma \mathbf{g}_t \tag{8}$$

where γ denotes learning rate. Updated parameters are sent back to each worker to compute the gradients for the $(t+1)$-th iteration [14, 23]. The process is repeated untiall convergence is attained.

4.2 Proposed distributed learning approach

Based on the motivation (see Section 2), we use QCS to compress the sparse gradients and obtain the quantized compressed measurement $\mathbf{y}_t^{(k)}$ as follows,

$$\mathbf{y}_t^{(k)} = Q(\mathbf{\Phi}^{(k)}\mathbf{g}_t^{(k)}) \tag{9}$$

Each worker sends the compressed measurements or $\mathbf{y}_t^{(k)}$ to the parameter server. As the quantization is performed on the compressed gradients, our approach requires lower communication cost over standard gradient quantization approaches (where quantization is performed directly on the gradients).

At the parameter server the quantized compressed measurements are recovered to obtain $\tilde{\mathbf{g}}_t^{(k)}$. Parameter server performs the gradient aggregation $\tilde{\mathbf{g}}_t = \frac{1}{K}\sum_{k=1}^{K}\tilde{\mathbf{g}}_t^{(k)}$ followed by the parameter update shown as,

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma \tilde{\mathbf{g}}_t \tag{10}$$

where γ denotes the learning rate.

Note: In this work, we focused on providing a convergence analysis for a general setup where each worker uses different measurement matrix (which is available to the parameter server a priori). As a result, parameter server needs to perform QCS recovery K times. Whereas, if each user uses same measurement matrix, parameter server would require to perform QCS recovery only once. Following measurement vector would be considered in the aforementioned case: $\mathbf{y}_t = Q(\mathbf{\Phi}[\frac{1}{K}\sum_{k=1}^{K}\mathbf{g}_t^{(k)}])$. Aggregated gradient $\tilde{\mathbf{g}}_t$ can be obtained by performing recovery only once, as opposed to K times. The convergence rate can be modified accordingly.

4.3 Convergence analysis

In this section we analyse the convergence of the proposed approach in the non-convex setting, which is typical in most of the deep learning systems. For this analysis we follow the standard assumptions of the stochastic optimization summarized by Allen et al. [1].

Assumption 1 *$\forall \mathbf{w}$ and some constant f^*, global objective function $f(\mathbf{w}) > f^*$.*

Above assumption guarantees the convergence of the global objective function to a stationary point.

Assumption 2 *Let $\bar{\mathbf{g}}(\mathbf{w})$ denote $\nabla\mathbf{f}(\mathbf{w})$ evaluated at $\mathbf{w} = [w_1, w_2, \ldots, w_d]^T$. Then $\forall \mathbf{w}$, $\mathbf{\Theta} = [\theta_1, \theta_2, \ldots, \theta_d]^T$ and a non-negative constant vector $\mathbf{L} = [l_1, l_2, \ldots, l_d]^T$,*

$$|f(\Theta) - [f(\mathbf{w}) + \bar{\mathbf{g}}(\mathbf{w})^T(\Theta - \mathbf{w})]| \leq \frac{1}{2}\sum_{i=1}^{d} l_i(\theta_i - w_i)^2$$

Above assumption acts as a smoothness criteria. We define $l' = ||\mathbf{L}||_\infty$.

Assumption 3 *Stochastic gradient* $\mathbf{g}(\mathbf{w})$ *is an unbiased estimate having bounded coordinate variance* $\mathbb{E}[\mathbf{g}(\mathbf{w})] = \bar{\mathbf{g}}(\mathbf{w})$ *and,*

$$\mathbb{E}[(\mathbf{g^{(k)}}(\mathbf{w})_i - \bar{\mathbf{g}}(\mathbf{w})_i)^2] \leq \sigma_i^2$$

for some non-negative constant vector $\boldsymbol{\sigma} = [\sigma_1, \sigma_2, \ldots, \sigma_d]^T$.

Assumption 4 *Let* $\bar{\mathbf{n}}_t = \mathbb{E}[\mathbf{n}_t]$ *and there exists a non-negative* μ *such that (for* $\mu < 1$*),*

$$||\bar{\mathbf{n}}_t|| \leq \mu ||\bar{\mathbf{g}}_t||$$

Under assumptions {1, 2, 3, 4} we have the following convergence rate:

Theorem 1 *Let* T *be the total number of iterations and learning rate* $\gamma = \frac{1}{l' K \sqrt{T}}$ *and* f_0 *be the initial objective value. Then,*

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T-1} ||\bar{\mathbf{g}}_t||^2\right] \leq \frac{1}{\sqrt{T}}\left[\frac{l' K^2 (f_0 - f^*) + ||\boldsymbol{\sigma}||^2 + \beta}{1-\mu}\right]$$

Proof. From assumption 2 we can write,

$$f_{t+1} - f_t \leq \bar{\mathbf{g}}_t^T (\mathbf{w}_{t+1} - \mathbf{w}_t) + \frac{1}{2}\sum_{i=1}^{d} l_i (\mathbf{w}_{t+1} - \mathbf{w}_t)_i^2 \tag{11}$$

where f_t denotes the global objective at t-th iteration and the gradient of which is denoted by $\bar{\mathbf{g}}_t$.

By taking the expected improvement conditioned on $\mathbf{w}_t$ we get,

$$\mathbb{E}[f_{t+1} - f_t | \mathbf{w}_t] \leq \overbrace{\mathbb{E}[\bar{\mathbf{g}}_t^T (\mathbf{w}_{t+1} - \mathbf{w}_t) | \mathbf{w}_t]}^{\mathbf{I}} + \overbrace{\mathbb{E}\left[\frac{1}{2}\sum_{i=1}^{d} l_i (\mathbf{w}_{t+1} - \mathbf{w}_t)_i^2 | \mathbf{w}_t\right]}^{\mathbf{II}} \tag{12}$$

Considering part $\mathbf{I}$ of equation 12 we can write,

$$\begin{aligned}
\mathbb{E}[\bar{\mathbf{g}}_t^T (\mathbf{w}_{t+1} - \mathbf{w}_t) | \mathbf{w}_t] &= -\mathbb{E}[\bar{\mathbf{g}}_t^T \gamma \frac{1}{K}\sum_{k=1}^{K} \tilde{\mathbf{g}}_t^{(k)} | \mathbf{w}_t] \\
&= -\gamma \bar{\mathbf{g}}_t^T \mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K} (\mathbf{g}_t^{(k)} - \mathbf{n}_t^{(k)}) | \mathbf{w}_t\right] \\
&= -\gamma \bar{\mathbf{g}}_t^T \mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K} \mathbf{g}_t^{(k)} | \mathbf{w}_t\right] + \gamma \bar{\mathbf{g}}_t^T \mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K} \mathbf{n}_t^{(k)} | \mathbf{w}_t\right] \\
&= -\gamma \bar{\mathbf{g}}_t^T \bar{\mathbf{g}}_t + \gamma \bar{\mathbf{g}}_t^T \bar{\mathbf{n}}_t
\end{aligned} \tag{13}$$

Considering Assumptions 3 and 4, Equation 13 can be simplified as below,

$$\begin{aligned}
\mathbb{E}[\bar{\mathbf{g}}_t^T (\mathbf{w}_{t+1} - \mathbf{w}_t) | \mathbf{w}_t] &\leq -\gamma ||\bar{\mathbf{g}}_t||^2 + \gamma ||\bar{\mathbf{g}}_t|| ||\bar{\mathbf{n}}_t|| \\
&\leq -\gamma ||\bar{\mathbf{g}}_t||^2 + \gamma \mu ||\bar{\mathbf{g}}_t||^2
\end{aligned} \tag{14}$$

Considering **II** of Equation 12,

$$\begin{aligned}\mathbb{E}\left[\frac{1}{2}\sum_{i=1}^{d} l_i(\mathbf{w}_{t+1}-\mathbf{w}_t)_i^2|\mathbf{w}_t\right] \leq & \mathbb{E}\left[\frac{1}{2}l'||\mathbf{w}_{t+1}-\mathbf{w}_t||^2|\mathbf{w}_t\right] \\ = & \mathbb{E}\left[\frac{1}{2}l'\left|\left|\frac{\gamma}{K}\sum_{k=1}^{K}\tilde{\mathbf{g}}_t^{(k)}\right|\right|^2\middle|\mathbf{w}_t\right] \\ \leq & \frac{l'\gamma^2}{2K}\sum_{k=1}^{K}\mathbb{E}[||\tilde{\mathbf{g}}_t^{(k)}||^2|\mathbf{w}_t] \\ = & \frac{l'\gamma^2}{2K}\sum_{k=1}^{K}\mathbb{E}[||\mathbf{g}_t^{(k)}-\mathbf{n}_t^{(k)}||^2|\mathbf{w}_t] \\ \leq & \frac{l'\gamma^2}{2K}\left[\sum_{k=1}^{K}2\mathbb{E}\left[||\mathbf{g}_t^{(k)}||^2|\mathbf{w}_t\right]+\sum_{k=1}^{K}2\mathbb{E}\left[||\mathbf{n}_t^{(k)}||^2|\mathbf{w}_t\right]\right]\end{aligned} \tag{15}$$

From the variance bound of Assumption 3 we can write,

$$\mathbb{E}\left[||\mathbf{g}_t^{(k)}-\bar{\mathbf{g}}_t||^2|\mathbf{w}_t\right] \leq ||\boldsymbol{\sigma}||^2 \tag{16}$$

Equation 16 can be re-written as,

$$\begin{aligned}||\boldsymbol{\sigma}||^2 \geq & \mathbb{E}\left[||\mathbf{g}_t^{(k)}-\bar{\mathbf{g}}_t||^2\right] \\ = & \mathbb{E}\left[||\mathbf{g}_t^{(k)}||^2-2\bar{\mathbf{g}}_t^T\mathbf{g}_t^{(k)}+||\bar{\mathbf{g}}_t||^2\right] \\ = & \mathbb{E}\left[||\mathbf{g}_t^{(k)}||^2\right]-2\bar{\mathbf{g}}_t^T\mathbb{E}\left[||\mathbf{g}_t^{(k)}||\right]+||\bar{\mathbf{g}}_t||^2 \\ = & \mathbb{E}\left[||\mathbf{g}_t^{(k)}||^2\right]-2\bar{\mathbf{g}}_t^T\bar{\mathbf{g}}_t+||\bar{\mathbf{g}}_t||^2 \\ = & \mathbb{E}\left[||\mathbf{g}_t^{(k)}||^2\right]-||\bar{\mathbf{g}}_t||^2\end{aligned} \tag{17}$$

From Equation 17 we get,

$$\mathbb{E}\left[||\mathbf{g}_t^{(k)}||^2\right] \leq ||\boldsymbol{\sigma}||^2+||\bar{\mathbf{g}}_t||^2 \tag{18}$$

Substituting Equations 7 and 18 into Equation 15 we can write,

$$\mathbb{E}\left[\frac{1}{2}\sum_{i=1}^{d} l_i(\mathbf{w}_{t+1}-\mathbf{w}_t)_i^2|\mathbf{w}_t\right] \leq \gamma^2 l'\left[||\boldsymbol{\sigma}||^2+||\bar{\mathbf{g}}_t||^2+\beta\right] \tag{19}$$

Combining Equation 14 and 19 Equation 12 can be simplified as shown below,

$$\mathbb{E}[f_{t+1}-f_t|\mathbf{w}_t] \leq -\gamma||\bar{\mathbf{g}}_t||^2+\gamma\mu||\bar{\mathbf{g}}_t||^2+\gamma^2 l'\left[||\boldsymbol{\sigma}||^2+||\bar{\mathbf{g}}_t||^2+\beta\right] \tag{20}$$

Substituting the value of γ in Equation 20,

$$\begin{aligned}\mathbb{E}[f_{t+1}-f_t|\mathbf{w}_t] \leq & ||\bar{\mathbf{g}}_t||^2\left(\frac{1}{l'TK^2}-\frac{1-\mu}{l'\sqrt{T}K}\right)+\frac{1}{l'TK^2}(||\boldsymbol{\sigma}||^2+\beta) \\ \leq & -\frac{1-\mu}{l'\sqrt{T}K^2}||\bar{\mathbf{g}}_t||^2+\frac{1}{l'TK^2}(||\boldsymbol{\sigma}||^2+\beta)\end{aligned}$$

Let us further extend the expectation over randomness in the trajectory and perform a telescoping sum over all the iterations. We obtain,

$$\begin{aligned}
f_0 - f^* \geq & f_0 - \mathbb{E}[f_T] \\
= & \mathbb{E}\left[\sum_{t=0}^{T-1}(f_t - f_{t+1})\right] \\
\geq & \frac{1}{l'}\mathbb{E}\sum_{t=0}^{T-1}\left[\frac{(1-\mu)||\bar{\mathbf{g}}_t||^2}{\sqrt{T}K^2} - \frac{||\boldsymbol{\sigma}||^2+\beta}{TK^2}\right] \\
= & \mathbb{E}\left[\frac{\sqrt{T}(1-\mu)}{l'K^2}||\bar{\mathbf{g}}_t||^2 - \frac{||\sigma||^2+\beta}{l'K^2}\right] \\
= & \frac{1}{l'K^2}\left\{\sqrt{T}(1-\mu)\mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T-1}||\bar{\mathbf{g}}_t||^2\right] - (||\sigma||^2+\beta)\right\}
\end{aligned}$$

By rearranging the above inequality we can write,

$$\mathbb{E}\left[\frac{1}{T}\sum_{t=0}^{T-1}||\bar{\mathbf{g}}_t||^2\right] \leq \frac{1}{\sqrt{T}}\left[\frac{l'K^2(f_0 - f^*) + ||\boldsymbol{\sigma}||^2 + \beta}{1-\mu}\right]$$

This completes the proof. ■

Note that the asymptotic convergence rate of the proposed approach is $\mathbf{O}\left(\frac{\beta}{\sqrt{T}}\right)$. In comparison, SGD has the same asymptotic convergence rate of $\mathbf{O}\left(\frac{\beta}{\sqrt{T}}\right)$. Earlier work on error-compensated DoubleSqueeze admits the same convergence rate of $\mathbf{O}\left(\frac{\beta}{\sqrt{T}}\right)$ [16].

5 Conclusion

In this work, we introduce a novel quantized compressive sensing-based gradient compression approach. We exploit the gradient sparsity induced by the activation layers in DNNs to reduce the communication cost between the workers and the parameter server in a distributed learning framework. We also provide a theoretical analysis on the convergence of the proposed approach.

Further studies would involve validation with experimental results. Comparison would be made against the state of the art quantization-based gradient compression technique. Suitability of a low complexity QCS recovery algorithm would be investigated. The work would further be extended into de-centralized setting exploiting joint sparsity.

6 Acknowledgement

We thank Nikhil Krishnan and Erfan Hosseini for interesting discussions and feedback in providing the convergence analysis.

References

[1] Zeyuan Allen-Zhu. Natasha: Faster non-convex stochastic optimization via strongly non-convex parameter. In ICML, 2017.

[2] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. Signsgd: Compressed optimisation for non-convex problems. ArXiv, abs/1802.04434, 2018.

[3] Petros Boufounos and Richard Baraniuk. 1-bit compressive sensing. In 42nd Annual Conference on Information Sciences and Systems, pages 16–21, March 2008.

[4] Emmanuel J. Candès. Theory of signals/mathematical analysis. Comptes rendus - Mathematique, 346(9-10):589–592, 2008.

[5] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Józefowicz. Revisiting distributed synchronous sgd. ArXiv, abs/1604.00981, 2017.

[6] D Donoho. Compressed sensing. IEEE Transactions on Information Theory, 52(4):1289–1306, April 2006.

[7] David Donoho. For most large underdetermined systems of linear equations the minimal ℓ_1-norm solution is also the sparsest solution. Comm. Pure Appl. Math, 59:797–829, 2004.

[8] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. CoRR, abs/1510.00149, 2015.

[9] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In Advances in Neural Information Processing Systems 29, pages 4107–4115. Curran Associates, Inc., 2016.

[10] Mu Li, David Andersen, Alexander Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In Advances in Neural Information Processing Systems 27, pages 19–27. Curran Associates, Inc., 2014.

[11] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In ICML, pages 807–814. Omnipress, 2010.

[12] Jongsoo Park, Sheng R. Li, Wei Wen, Ping Tak Peter Tang, Hai Li, Yiran Chen, and Pradeep Dubey. Faster cnns with direct sparse convolutions and guided pruning. In ICLR, 2016.

[13] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In Advances in Neural Information Processing Systems 24, pages 693–701. Curran Associates, Inc., 2011.

[14] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In CCS 2015 - Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pages 1310–1321, 10 2015.

[15] Sebastian Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified SGD with memory. In Advances in Neural Information Processing Systems 31, pages 4447–4458. Curran Associates, Inc., 2018.

[16] Hanlin Tang, Chen Yu, Xiangru Lian, Tong Zhang, and Ji Liu. `DoubleSqueeze`: Parallel stochastic gradient descent with double-pass error-compensated compression. In Proceedings of the 36th International Conference on Machine Learning, volume 97, pages 6155–6165, Long Beach, California, USA, 09–15 Jun 2019.

[17] Albericio et al. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), pages 1–13, June 2016.

[18] Chen et al. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. ArXiv, abs/1512.01274, 2015.

[19] Dan et al. The convergence of sparsified gradient methods. In Advances in Neural Information Processing Systems 31, pages 5973–5983. Curran Associates, Inc., 2018.

[20] Ivkin et al. Communication-efficient distributed sgd with sketching. In NeurIPS, 2019.

[21] Li et al. Scaling distributed machine learning with the parameter server. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pages 583–598, Broomfield, CO, October 2014. USENIX Association.

[22] Martín et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. ArXiv, abs/1603.04467, 2015.

[23] Wei et al. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In Advances in Neural Information Processing Systems 30, pages 1509–1519. Curran Associates, Inc., 2017.

[24] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In Advances in Neural Information Processing Systems 29, pages 2074–2082. Curran Associates, Inc., 2016.

[25] X. Yao, C. Huang, and L. Sun. Two-stream federated learning: Reduce the communication costs. In 2018 IEEE Visual Communications and Image Processing (VCIP), pages 1–4, Dec 2018.

[26] Xin Yao, Chaofeng Huang, and Lifeng Sun. Two-stream federated learning: Reduce the communication costs. 2018 IEEE Visual Communications and Image Processing (VCIP), pages 1–4, 2018.

[27] Xin Yao, Tianchi Huang, Chenglei Wu, Rui-Xiao Zhang, and Lifeng Sun. Federated learning with additional mechanisms on clients to reduce communication costs. ArXiv, abs/1908.05891, 2019.