SODA: SELF-ORGANIZING DATA AUGMENTATION IN DEEP NEURAL NETWORKS - APPLICATION TO BIOMEDICAL IMAGE SEGMENTATION TASKS

Arnaud Deleruyelle, John Klein, Cristian Versari

Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL, F-59000 Lille

ABSTRACT

In practice, data augmentation is assigned a predefined budget in terms of newly created samples per epoch. When using several types of data augmentation, the budget is usually uniformly distributed over the set of augmentations but one can wonder if this budget should not be allocated to each type in a more efficient way. This paper leverages online learning to allocate on the fly this budget as part of neural network training. This meta-algorithm can be run at almost no extra cost as it exploits gradient based signals to determine which type of data augmentation should be preferred. Experiments suggest that this strategy can save computation time and thus goes in the way of greener machine learning practices.

Index Terms— data augmentation, HEDGE, online learning, deep learning, segmentation

1. INTRODUCTION

The benefits of data augmentation (DA) to train deep neural networks has been widely acknowledged. This technique consists in applying various transformations to training examples in order to obtain new ones. Provided that the data distribution is invariant with respect to these transformations, the newly created samples help learning a model with better generalization performances because one is learning from a larger training set. Even when the distribution is not invariant with respect to the transformations, DA can also be beneficial and prevent from overfitting the original training dataset.

There are many possibilities for machine learning practitioners to generate augmented data. We focus in this paper on deep neural network models whose inputs are images. The most popular DA transformations for images include rotations, flipping, noise injection, color or illumination changes, or non-rigid deformations among others. More recently, there has been attempts to use previously trained generative models to sample new data [1, 2]. Such models can also create new samples through style transfer [3]. Another line of thought is to learn how to augment in the spirit of *learning-to-learn* meta-learning algorithms [4, 5]. Such approaches usually involve a pair of networks (one to solve the main task and one to learn to augment data meaningfully). Note that data augmentations can be combined, which makes the number of possible choices very large. For a review of data augmentation techniques, the reader is referred to [6].

Regardless of the nature of data augmentation types, in practice, one has to (i) choose a limited number of them and (ii) define a budget in terms of newly created samples through augmentation. Sampling too many augmented data might be counter-productive in terms of exploitation of computational resources. In general, it is however hard to anticipate which DA type should be chosen. In addition, the appeal of a given DA type might not be constant across training epochs, i.e. the problem is not a mere scalar hyperparameter setting issue but involves determining a whole sequence of actions.

This paper investigates a mean of learning on the fly which DA types should be preferred through already computed gradients as part of neural network training through gradient based optimizers. Because we exploit gradient signals, our approach has almost no extra cost compared to backpropagation. Building upon HEDGE, an online learning algorithm, we design an algorithm that adaptively determines a score for each DA type and reduces/increases the number of samples obtained from each of them for the next epoch accordingly.

The closest related work with respect to our contribution is the AutoAugment approach from Cubuk et al. [7]. This approach uses reinforcement learning (RL) to find an efficient sequence of DA types and magnitudes for each batch. The RL procedure requires to re-train a neural network and evaluate its accuracy in order to obtain a feedback that allows converging to a meaningful policy. This is in sharp contrast with the proposed approach in which the neural network is trained only once and the sequence of DA types is determined on the fly. While the number of possible sequences can be extremely large, the problem can still be regarded as a hyperparameter optimization one. But again, techniques such as Bayesian Optimization or Multi-Armed Bandits (MABs) [8] from the literature require to evaluate a sequence through repeated training runs of the chosen neural network architecture.

The paper is organized as follows. The next section formalizes the DA allocation problem and presents the HEDGE framework. Section 3 provides a detailed presentation of our contribution whose backbone is HEDGE. The proposed algorithm is evaluated on U-Net [9] architectures for segmentation tasks on biomedical images. Experimental material is

presented in section 4 and suggests that our approach allows saving computation time compared to naive policies while achieving comparable accuracy.

2. PROBLEM STATEMENT AND BACKGROUND

2.1. Preliminaries

In the supervised learning setting, one has access to a training dataset $\mathcal{D}$ which contains n pairs $(\mathbf{x}, y)$ of inputs/targets. Given a parametric model such as a neural network $f_{\boldsymbol{\theta}}$, our task consists in determining the best vector of trainable parameters $\boldsymbol{\theta}$ such that, for each training example, $L\left(f_{\boldsymbol{\theta}}\left(\mathbf{x}\right), y\right)$ is small where L is the chosen loss function. If n is large enough, the training algorithm issuing the model minimizes the expected loss (in a probably approximately correct sense).

Now, suppose one can afford to learn from n_a additional augmented training examples per epoch and has K DA generators to choose from to allocate this computational budget. Given a K dimensional vector $\boldsymbol{\pi}$ of probabilities, a π_k fraction of the n_a augmented examples is queried to the DA generator k. In the current paper, we are interested in estimating a meaningful vector $\boldsymbol{\pi}$. Because the appropriate proportion of each type of DA is not constant over training epochs, we will use an index t for this vector in the sequel. Moreover, we assume $n_a \gg K$ so that each DA receives at least a budget of one point.

2.2. Online learning with HEDGE

Let us further assume that the neural network training environment provides a K-dimensional bounded action-loss signal $\boldsymbol{\ell}_t$ that tells us how much each DA type is unhelpful in training our model. This setting is an online optimization problem of learning with expert advice where the learner suffers a loss given by the following dot product $\boldsymbol{\pi}_t^{\intercal} \cdot \boldsymbol{\ell}_t$ (at each time step). HEDGE [10] is an algorithm achieving minimal regret w.r.t. the cumulative loss $\sum_{t=1}^{T} \boldsymbol{\pi}_t^{\intercal} \cdot \boldsymbol{\ell}_t$ [11] where T is the final epoch (assumed to be fixed here for simplicity).

HEDGE is also known as the aggregating algorithm or exponentially weighted forecaster. Unlike MABs which receive a feedback only for the previously chosen action, HEDGE receives a feedback for all actions. In our context, at each epoch t, HEDGE consists in the following sequence of steps:

1. Learner chooses $\boldsymbol{\pi}_t$, where $\pi_{k,t} = \frac{w_{k,t}}{\sum_{k=1}^{K} w_{k,t}}$.
2. Environment reveals $\boldsymbol{\ell}_t$.
3. Learner updates weights $w_{k+1,t} = w_{k,t} \exp\left(-\eta \ell_{k,t}\right)$ for each k.

In the last step, η is HEDGE learning rate. The weights characterize how helpful each DA type is. Before the first epoch, they are initialized as $w_{k,1} = 1, \forall k$. HEDGE can be employed in a context where the environment is adversarial (and chooses $\boldsymbol{\ell}_t$ to maximize regret). In this paper, the environment does not exhibit such a behavior, but may evolve with respect to t and thus we need to elaborate on this algorithmic procedure to fulfill our goal, as explained in the next section. It should be underlined that HEDGE has the advantage of not requiring action-loss distribution assumptions.

3. A NEW SELF-ORGANIZED DATA AUGMENTATION ALLOCATION ALGORITHM

This section introduces a DA allocation algorithm that can be coupled with neural network optimization at very small extra-cost. HEDGE is the backbone of this algorithm. Building upon HEDGE, we provide a workable and contextualized algorithm for DA allocation deployed in only one training run, in contrast with typical MAB based or offline allocations which use multiple runs.

3.1. Crafting a meaningful feedback signal

As part of HEDGE, the environment (the neural network training procedure in our case) should reveal at each epoch t how each option (DA type in our case) is unhelpful through loss values. In our context, and among other possibilities, the k^{th} action-loss is defined as the average train-loss discrepancy obtained by learning from this type of DA:

$$\frac{1}{n_{k,t}} \sum_{i=1}^{n_{k,t}} \frac{1}{n} \sum_{(\mathbf{x},y)\in\mathcal{D}} L\left(f_{\boldsymbol{\theta}+\Delta\boldsymbol{\theta}_{t,i}}\left(\mathbf{x}\right), y\right) - L\left(f_{\boldsymbol{\theta}}\left(\mathbf{x}\right), y\right), \tag{1}$$

where $n_{k,t}$ is the number of examples allocated to the k^{th} DA at epoch t and $\Delta\boldsymbol{\theta}_{t,i}$ is the parameter update obtained from the i^{th} augmented training example in this DA category. Note that $n_{k,t} \geq 1$ is a positive integer obtained by applying a rounding function to $\pi_{k,t} \times n_a$ so that $\sum_k n_{k,t} = n_a$.

It would be computationally prohibitive to evaluate the train-loss discrepancy for each augmented training example (n forward passes per augmented sample). To circumvent this issue, let us view the train-loss as a function J of $\boldsymbol{\theta}$. Using first order Taylor expansion, we have $J\left(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_{t,i}\right) \approx J\left(\boldsymbol{\theta}\right) + \nabla J\left(\boldsymbol{\theta}\right)^{\intercal} \cdot \Delta\boldsymbol{\theta}_{t,i}$. Moreover, assuming the neural network is optimized by stochastic gradient descent (SGD), then $\Delta\boldsymbol{\theta}_{t,i} = -\alpha \mathbf{g}_{t,i}^{(k)}$ where $\mathbf{g}_{t,i}^{(k)}$ is the gradient computed from the augmented training example and α is SGD learning rate. Consequently, the train-loss discrepancy can be approximated by a dot product of gradients.

The gradient $\nabla J\left(\boldsymbol{\theta}\right)$ remains prohibitive to compute. But, alleging that during one epoch the network will not evolve too much[1], this gradient can be replaced by the average gradient $\mathbf{g}_t^{(0)}$ obtained from data points in $\mathcal{D}$ throughout SGD execution. This is actually the idea behind SGD. Consequently,

[1] This holds for a sufficiently small α which is also required for the Taylor approximation to be good.

$-\alpha\ \mathbf{g}_t^{(0)\intercal} \cdot \mathbf{g}_t^{(k)}$ can be used as proxy for (1) where $\mathbf{g}_t^{(k)}$ is the average gradient obtained from augmented data points from the k^{th} DA throughout SGD execution in the current epoch. Finally, to comply with HEDGE boundedness requirements, we will use the following normalization based on the cosine of the angle between the gradients (which also removes constant α):

$$\ell_{k,t} = \frac{1}{2}\left(1 - \frac{\mathbf{g}_t^{(0)\intercal} \cdot \mathbf{g}_t^{(k)}}{\left\|\mathbf{g}_t^{(0)}\right\|_2 \left\|\mathbf{g}_t^{(k)}\right\|_2}\right), \tag{2}$$

where $\|.\|_2$ is the Euclidean $\mathcal{L}_2$ norm. The action-loss (2) is easy to interpret: it is a gradient matching based signal which has been also used in offline DA policy allocation learning [12, 13].

In order to work with more stable action-loss signals and mitigate gradient direction oscillations, we will use momentum estimates of average gradients, i.e. $\ell_{k,t} = \frac{1}{2}\left(1 - \frac{\tilde{\mathbf{g}}_t^{(0)\intercal} \cdot \tilde{\mathbf{g}}_t^{(k)}}{\left\|\tilde{\mathbf{g}}_t^{(0)}\right\|_2 \left\|\tilde{\mathbf{g}}_t^{(k)}\right\|_2}\right)$ where, for any $k \in 0, .., K$,

$$\mathbf{m}_t^{(k)} = \rho\, \mathbf{m}_{t-1}^{(k)} + (1-\rho)\, \mathbf{g}_t^{(k)}, \tag{3}$$

$$\tilde{\mathbf{g}}_t^{(k)} = \frac{\mathbf{m}_t^{(k)}}{1-\rho^t}, \tag{4}$$

and $\mathbf{m}_t^{(k)}$ is a momentum vector for the k^{th} DA which is initialized as $\mathbf{m}_t^{(k)} = \mathbf{0}$.

3.2. Online learning with a discount factor

Now that we have action-loss signals, one could apply HEDGE to solve our DA allocation task. However, as mentioned before, appropriate allocations may vary across epochs. To take into account this time-dependence, we propose to add a discount factor $\beta \in [0; 1]$ in the weight update of HEDGE, that is, step 3. from HEDGE is replaced with

$$w_{k+1,t} = w_{k,t}^{\beta} \exp\left(-\eta \ell_{k,t}\right), \forall k. \tag{5}$$

When $\beta = 1$, we retrieve HEDGE while when $\beta = 0$ we obtain a memoryless procedure. This simple modification of HEDGE allows the DA allocation algorithm to forget about past observed action-losses in order to more rapidly adapt to a shift of DA usefulness over epochs. Unrolling this update rule, a DA weight can be re-written as

$$w_{k,t} = \exp\left(-\eta \sum_{t'=1}^{t} \beta^{t-t'} \ell_{k,t'}\right). \tag{6}$$

We thus see that this variant of HEDGE is meant to minimize a discounted version of the cumulative loss $\sum_{t=1}^{T} \beta^{T-t} \boldsymbol{\pi}_t^{\intercal} \cdot \boldsymbol{\ell}_t$.

In total, there are three hyperparameters to tune (η, ρ and β) for our DA allocation algorithm which we call SODA for Self-Organizing Data Augmentation.

4. EXPERIMENTAL VALIDATION THROUGH BIOMEDICAL IMAGE SEGMENTATION

This section provides numerical experiments to evaluate the benefits of SODA. We chose a validation framework that uses U-Net architectures [9] on segmentation tasks. Indeed, in this kind of supervised problems, DA appears to be a critical aspect of the training process so that U-Net can generalize efficiently.

4.1. Datasets & Preprocessing

We evaluate our strategy on 4 datasets. The first dataset is DRIVE [14]. It contains the segmentation of blood vessels in retinal images. In the second one [15], a transparent capsule crossing in a micro-tube must be segmented. The last two datasets are 'DIC-C2DH-HeLa' and 'PhC-C2DH-U373' from the ISBI cell tracking challenge [16].

We only use $n = 20$ training examples for each experiment so that DA has a greater impact. Inputs are standardized (zero mean and unit variance). We use 3 DA types:

- noise injection: choose σ uniformly at random in $\{0.01, 0.02, \ldots, 0.05\}$ and multiply the image by a noise whose pixels are sampled from $\mathcal{N}(0, \sigma^2)$,
- rotation: choose a uniformly at random in $\{1, 2, \ldots, 8\}$ and apply a rotation of $a\frac{\pi}{4}$ to the image,
- "junk DA": replace the inputs with random images where each pixel is drawn uniformly in $[0, 1]$.

Obviously, the third DA should slow down the learning, while the remaining ones are expected to improve it.

4.2. Experiments

The tested Unet is implemented using Keras with Tensorflow backend and has the same architecture as [9] (8 layers, 3×3 filters + ReLU). However, we use zero-padding so that output shapes match input ones. The layers contain (from 1st to last) $10, 20, 40, 80, 80, 40, 20$ and 10 filters respectively. Except for the last layer, we use an $\mathcal{L}_2$ regularization penalty with regularization factor $1e-4$. The neural network optimizer is RMSProp with learning rate $1e-4$ (other parameters are left to default Keras values) which minimizes quadratic train-loss $L\left(f_{\boldsymbol{\theta}}\left(\mathbf{x}\right), y\right) = \frac{1}{n}\sum_{(\mathbf{x},y)\in\mathcal{D}} \left\|f_{\boldsymbol{\theta}}\left(\mathbf{x}\right) - y\right\|_2^2$.

To assess the performances of our DA allocation algorithm, we train U-Net 10 times and report average Jaccard index computed on a test set (disjoint from the training set) containing 30 samples. The Jaccard index is the number of pixels in the intersection of the predicted mask region and ground-truth (y) divided by the number of pixels in the union of the predicted and true masks. The predicted mask is obtained by comparing $f_{\boldsymbol{\theta}}(\mathbf{x})$ to a threshold of 0.5. The number of augmented data per epoch is $n_a = 60$. We compare SODA with

2 concurrent strategies: a uniform allocation over the 3 DA types and a "target" allocation which ignores the junk DA and evenly queries the remaining two. Ideally, SODA should be able to automatically discard the junk DA and achieve comparable performances without any prior knowledge on the usefulness of each DA type.

For each dataset, Table 1 gives SODA parameters, while Figure 1 shows the Jaccard index over epochs achieved by the different strategies. We can see that SODA outperforms the uniform allocation, regardless of the dataset. Its performances are also very close to the target strategy. In the last epochs, all strategies generally converge to close Jaccard index values. SODA learns at a faster pace which is useful in strongly computationally constrained learning environments where only a small number of epochs are possible.

Dataset	η	ρ	β
DRIVE database	6	0.99	0.5
Capsule	4	0.99	0.5
ISBI: DIC-C2DH-HeLa	3	0.99	0.5
ISBI: PhC-C2DH-U373	7	0.99	0.5

Table 1: SODA hyperparameters for each dataset. We only vary η and keep ρ and β to the same value for all datasets which shows that hyperparemeter setting can essentially focus on SODA learning rate.

5. CONCLUSION

This paper introduces SODA, an algorithm for Self-Organizing Data Augmentation that can be deployed within usual deep neural network training loops. This algorithm leverages online learning to identify relevant allocation of computational resources for each type of data augmentation. We show that the benefits of each type of data augmentation can be monitored through the dot product of two average gradients: the one from the data augmentation itself, and the one obtained from data points belonging to the original training set. To account for the time variability of meaningful allocations, a discounted version of the algorithm is used.

In our experiments on segmentation in biomedical images using a U-Net architecture, SODA proves to be able to outperform a standard uniform distribution of computational resources with respect to data augmentation types. According to our results, SODA achieves comparable accuracy w.r.t. the uniform policy but at a faster rate, which may be helpful for reducing ML carbon impact.

In future works, we wish to generalize SODA to situations where there is no relevant data augmentation to choose from. While SODA exhibits improved robustness, we also believe the action-loss signals could be further processed in order to discard irrelevant data augmentation types more easily.

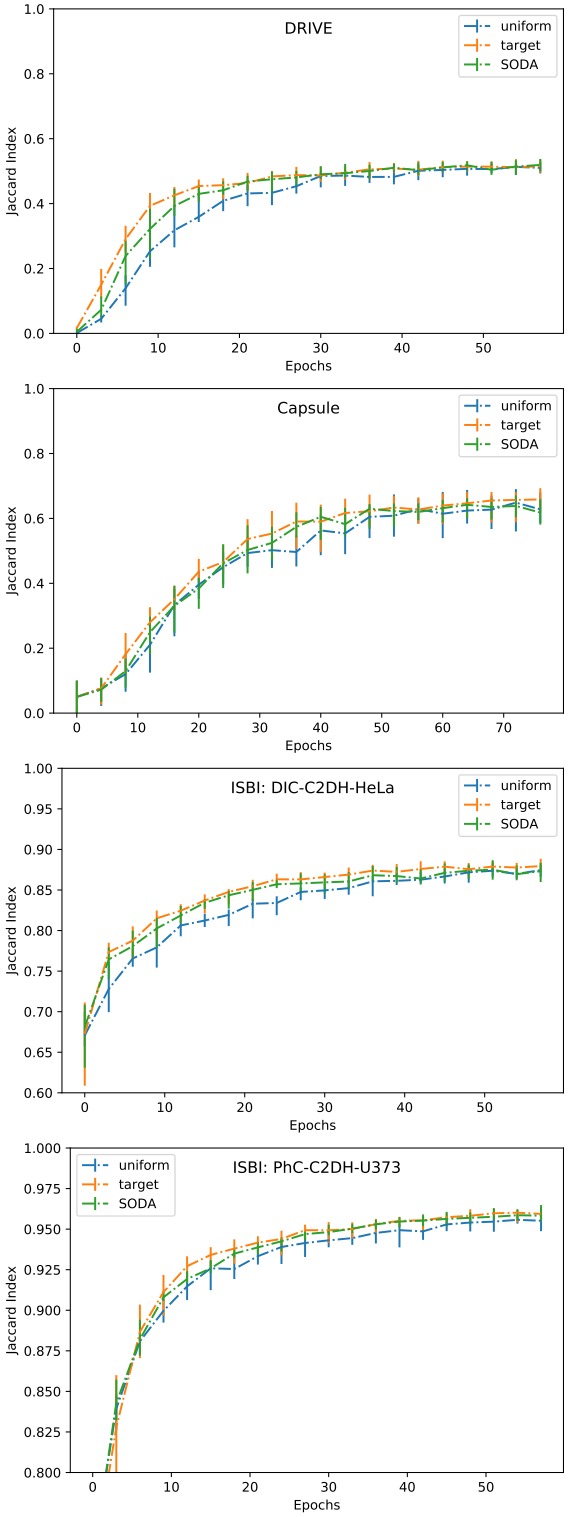

Fig. 1: Accuracy w.r.t to epochs for the 4 datasets and the 3 allocation strategies.

Acknowledgements: This work was performed using HPC resources from GENCI–IDRIS (Grant 2021-AD011011606R1).

6. REFERENCES

[1] Xinyue Zhu, Yifan Liu, Jiahong Li, Tao Wan, and Zengchang Qin, "Emotion classification with data augmentation using generative adversarial networks," in *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 2018, pp. 349–360.

[2] Michal Amitai Jacob Goldberger Hayit Greenspan Maayan Frid-Adar, Eyal Klang, "Synthetic data augmentation using gan for improved liver lesion classification," in *International Symposium on Biomedical Imaging (ISBI 2018)*, Washington D.C., USA, 2018.

[3] Philip TG Jackson, Amir Atapour Abarghouei, Stephen Bonner, Toby P Breckon, and Boguslaw Obara, "Style augmentation: data augmentation via style randomization.," in *CVPR Workshops*, 2019, pp. 83–92.

[4] Jason Wang, Luis Perez, et al., "The effectiveness of data augmentation in image classification using deep learning," *Convolutional Neural Networks Vis. Recognit*, vol. 11, pp. 1–8, 2017.

[5] Hiroshi Inoue, "Data augmentation by pairing samples for images classification," *arXiv preprint arXiv:1801.02929*, 2018.

[6] Connor Shorten and Taghi M Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.

[7] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le, "Autoaugment: Learning augmentation strategies from data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[8] Yimin Huang, Yujun Li, Hanrong Ye, Zhenguo Li, and Zhihua Zhang, "An asymptotically optimal multi-armed bandit algorithm and hyperparameter optimization," *arXiv preprint arXiv:2007.05670*, 2020.

[9] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2015, pp. 234–241.

[10] Yoav Freund and Robert E Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[11] Nicolo Cesa-Bianchi and Gábor Lugosi, *Prediction, learning, and games*, Cambridge university press, 2006.

[12] Aoming Liu, Zehao Huang, Zhiwu Huang, and Naiyan Wang, "Direct differentiable augmentation search," in *IEEE Int. Conf. on Computer Vision (ICCV'21)*, 2021, pp. 12219–12228.

[13] Yu Zheng, Zhi Zhang, Shen Yan, and Mi Zhang, "Deep autoaugment," *Transformation*, vol. 3, pp. 3, 2021.

[14] Joes Staal, Michael D Abràmoff, Meindert Niemeijer, Max A Viergever, and Bram Van Ginneken, "Ridge-based vessel segmentation in color images of the retina," *IEEE transactions on medical imaging*, vol. 23, no. 4, pp. 501–509, 2004.

[15] Anne-Virginie Salsac, Arnaud Deleruyelle, Cristian Versari, and John Klein, "Capsule: a dataset for the segmentation of a transparent and deformable capsule.," Online at: `https://github.com/ArnaudDeleruyelle/Dataset_Capsule`, 2021.

[16] Martin Maška, Vladimír Ulman, David Svoboda, Pavel Matula, Petr Matula, Cristina Ederra, Ainhoa Urbiola, Tomás España, Subramanian Venkatesan, Deepak MW Balak, et al., "A benchmark for comparison of cell tracking algorithms," *Bioinformatics*, vol. 30, no. 11, pp. 1609–1617, 2014.