MLP Only Batchnorm

December 22, 2022

Antonio Soldano

1. Introduction

The aim of the present project is to discuss and test a phenomena happening while training a neural network built with batch normalization. In the paper (Frankle et al., 2020) taken as main route for the focus of the project it has been found out that a CNN with weights frozen at their random original value and trained only on the β and γ parameters of the batch-normalization can achieve surprisingly good results on image classification problems. The following sections explain why it happens and show the process it has been undertaken to test experimentally this result on other architectures and other kinds of data.

2. Related work

The work carried out in (Frankle et al., 2020) is based on the effectiveness in improving the results of a neural network by using the affine parameters β and γ of the batch normalization. In order to study these parameters in isolation, the work has performed experiments on networks freezing all weights at their initialization value and training only the beta and gamma parameters in BatchNorm. Since the features in the network are fixed at random initializations, the affine parameters can only shift and scale (respectively β and γ) the normalized activation maps that these features produce in each layer.
It has been found out that, training over only batch normalization parameters, the accuracy of the network reaches surprisingly high results. The reason for this lies on the fact that the parameters are themselves trainable, so the experiment can be seen as training a ”smaller” neural network i.e. with fewer parameters and supported only by the shifts and rescalings of random features. Besides this, another positive side is the reduction of the *Internal Covariate Shift*: *the change in the distribution of network activations due to the change in network parameters during training*(Ioffe & Szegedy, 2015). The data used for this project comes from malware samples, more precisely executable (image) files and object files under the Windows family of operating systems, referred to as **Portable Executable (PE)**. Two related works have been taken as reference:

1. *Transfer Learning for Image-Based Malware Classification* (Bhodia et al., 2019): the authors have used a dataset of malware samples, extracted PE headers values and built gray-scale images generated from malware executables. Then, they applied image recognition and the results have been compared with a k-nearest neighbors (k-NN) technique. Surprisingly, the former is outperformed by k-NN.

2. *Malware Classification using classical Machine Learning and Deep Learning* (Prajapati & Stamp, 2021): is an extention of the previous one and part of the code, available on Github (Prajapati & Stamp), has been used in the current project. I personally consider this as the most precise and extensive one among the related works I could find. They used multiple learning techniques to solve the same task (malware classification) such as a **Multilayer Perceptron**, a **Convolutional Neural Netowrk** and a **Recurrent Neural Network**.

3. Project presentation

For the developement of the project two datasets have been used.

Dike Dataset. The first dataset used for this work is the one taken from a Github repository (Iosif). and it has been also used for similar purposes (Wei et al., 2021). In this case the task is binary classification, identifying the different executable files as *malware* or *benign*.

The Zoo Dataset. The second dataset used is taken from another well maintaned Github repository (tisf Nativ & Shalev.). It contains live and dangerous malware samples, in fact they come encrypted and locked because of this. In this case the task is multiple class classification, identifying each sample with the corresponding type of malware with a total of 8 output classes.

Email: Antonio Soldano <antonio.soldano@studenti.uniroma1.it>.

Deep Learning and Applied AI 2022, Sapienza University of Rome, 2nd semester a.y. 2021/2022.

Architecture. The built architectures are inspired on a paper in which malwares are analysed using Windows exe API Calls (Catak et al., 2020) The experiments have been carried out on two MLPs: one composed by 3 linear layers and batch normalization after all the layers and the other one with 7 linear layers and batch normalization as well.

Code and repository The code developed for this project is shared on Github (Soldano, 2022a): it has been fully written in Python making use of the Pythorch library and Streamlit, to let it be interactive. Moreover, Wandb has been used to log the results of the runs (Soldano, 2022b).

4. Results

Experiments were launched with different parameters including epochs, batch size, learning rate, optimizer and momentum (if any) to compare how they affected accuracy.

Table 1. Runs on Dike Dataset

Run Name	Epochs	Batch Size	Learning Rate	Optimizer
R1	15	32	0.001	SGD
R2	15	32	0.1	SGD
R3	30	64	0.001	SGD
R4	15	32	0.1	Adagrad
R5	50	32	0.1	Adagrad

Dike dataset results Quite all the runs reach a very high accuracy with respect to both training and validation. Altough it usually indicates that the network has overfit the data, achieving high accuracy on the validation set is a good sign that the network is able to generalize to new, unseen data. In general, we can see that the batch size and the number of epochs do not affect significantly the results (R1 and R3) as, instead, the learning rate and the choice of the optimizer do. Indeed, by increasing the learning rate mantaining the same hyperparameters increases the accuracy from R1 to R2 and does it remarkably between the same runs training over only batch normalization parameters (R1_Only_Batchnorm and R2_Only_Batchnorm). Furthermore, changing the optimizer lets the network reach perfect results, either in the normal runs or in the only batchnorm ones (R4, R5 and related Only_Batchnorms). At the end, increasing the momentum value does not change the accuracy but only the oscillation towards the steepest ascent direction of the loss.

The Zoo Dataset results In this case, the maximum accuracy reached by the network goes aroun 65% in run R5 during training and 62% during validation in run R2. The accuracy has strongly decreased compared with the previous dataset due to the higher number of classes the network has been trained on. As it has been highlighted, there is a clear positive correlation between learning rate and accuracy. This suggests that increasing the learning rate can lead to better performance for this particular neural network and dataset. The phenomenon is even more apparent when we compare the respective only BatchNorm runs. For this dataset it has been added another run, the R2.5 one with same hyperparameters of the R2 but higher learning rate and this has reached the same accuracy of the previous one. In general, setting the learning rate too high can cause the algorithm to diverge and fail to converge, while setting it too low can lead to slow training. In this case, it was not useful to improve the performances. As told in the previous paragraph, also changing the Optimizer leads to better performances. Moreover, it may be a good improvement to add weight decay or dropout to the architecture. Anyway, the batch normalisation itself used in the project has a regularisation effect so I personally believe that by adding regularisation techniques might not improve the results significantly.

Table 2. Runs on The Zoo Dataset

Run Name	Epochs	Batch Size	Learning Rate	Optimizer
R1	30	32	0.001	SGD
R2	30	32	0.1	SGD
R2dot5	30	32	0.5	SGD
R3	30	64	0.001	SGD
R4	15	32	0.1	Adamax
R5	50	64	0.1	Adamax

5. Further considerations and Conclusion

In the related works it has not been reported a key point represented by cases R4 in architecture 1 and R2 (and R2.5) in architecture 2: the only batchnorm runs have reached a better accuracy on the validation set rather than the normal one. There could be two reasons for that:

- **Overfitting**: If the MLP trained on all of the parameters is overfitting the training data, it may not generalize well to the validation set, resulting in a lower accuracy. On the other hand, the MLP trained only on the batch normalization parameters may be less prone to overfitting and may perform better on the validation set as a result.
- **Regularization**: The batch normalization layers in the MLP may be acting as a form of regularization, helping to prevent overfitting and improve the generalization performance of the model.

From this we can derive a final consideration: by increasing the learning rate (and possibly also the optimiser) a neural network trailed only on batch normalization parameters achieves results as good as the same network trained over all the parameters with a lower learning rate.

References

Bhodia, N., Prajapati, P., Troia, F. D., and Stamp, M. Transfer learning for image-based malware classification. *CoRR*, abs/1903.11551, 2019. URL http://arxiv.org/abs/1903.11551.

Catak, F. O., Yazı, A. F., Elezaj, O., and Ahmed, J. Deep learning based sequential model for malware analysis using windows exe api calls. *PeerJ Computer Science*, 6:e285, July 2020. ISSN 2376-5992. doi: 10.7717/peerj-cs.285. URL https://doi.org/10.7717/peerj-cs.285.

Frankle, J., Schwab, D. J., and Morcos, A. S. Training batchnorm and only batchnorm: On the expressive power of random features in cnns. *CoRR*, abs/2003.00152, 2020. URL https://arxiv.org/abs/2003.00152.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

Iosif, G.-A. Github repository - dike dataset. URL https://github.com/iosifache/DikeDataset.

Prajapati, P. and Stamp, M. Github repository - an empirical analysis of image-based learning techniques for malware classification. URL https://github.com/pratikpv/malware_detect2/blob/d12d3918c6e7b7f5d147c59f2211efdccd7c230d/data_utils/extract_pe_features.py.

Prajapati, P. and Stamp, M. *An Empirical Analysis of Image-Based Learning Techniques for Malware Classification*, pp. 411–435. Springer International Publishing, Cham, 2021. ISBN 978-3-030-62582-5. doi: 10.1007/978-3-030-62582-5_16. URL https://doi.org/10.1007/978-3-030-62582-5_16.

Soldano, A. Github project, 2022a. URL https://github.com/antoniosoldano/MalwareMLPBatchNorm.

Soldano, A. Wandb project, 2022b. URL https://wandb.ai/antoniosoldano/MLP%20Only%20Batchnorm.

tisf Nativ, Y. and Shalev., S. Github repository - the zoo dataset. URL https://github.com/ytisf/theZoo.

Wei, C., Li, Q., Guo, D., and Meng, X. Toward identifying apt malware through api system calls. *Security and Communication Networks*, vol. 2021, Article ID 8077220, 2021. URL https://doi.org/10.1155/2021/8077220.