WITC 2024
The 10th International Conference on Big data, IoT, and Cloud Computing

Scalability analysis of DeepVATS: A Framework for Deep Learning Visual Analytics on Large Time Series Data

Inmaculada Santamaria-Valenzuela[1]*, Víctor Rodriguez-Fernandez[1], Jong Hyuk Park[2], David Camacho[1]*

[1]Department of Computer Systems, Universidad Politécnica de Madrid, Spain

[2]Department of Computer Science and Engineering, Seoul National University of Science and Technology, (SeoulTech), Seoul 01811, Korea

mi.santamaria@upm.es, victor.rfernandez@upm.es, jhpark1@seoultech.ac.kr, david.camacho@upm.es

Abstract. DeepVATS is a Deep learning Visual Analytic tool that seamlessly integrates Deep Learning and Visual Analytics for the analysis of extensive time series data. This paper presents the main components of DeepVATS and performs a scalability analysis in terms of the complexity of the time series studied. Execution and rendering times have been logged and analysed to make scalability analysis.

Keywords: DeepVATS, Visual Analytics, scalability analysis, shiny

1 Introduction

DeepVATS is a tool inspired by TimeCluster [1], integrates Visual analytics (VA) and Deep Learning (DL). It uses dimensionally reduction (DR) techniques for efficient analysis and interaction with large time series. Its main tasks are: training neural networks to obtain its embeddings; projecting them in 2 dimensions to detect clusters and anomalies; and finally, providing interactive visualizations to explore different perspectives of the projected embeddings. This paper analyses DeepVATS' scalability

* Corresponding Author: Inmaculada Santamaria-Valenzuela (mi.santamaria@upm.es)

**Acknowledgement: Acknowledgement: This research was supported by Strategic Networking & Development Program funded by the Ministry of Science and ICT through the National Research Foundation of Korea (RS-2023-00267476), by Grant PLEC2021-007681 (XAI-DisInfodemics) and PID2020-117263GB-100 (FightDIS) funded by MCIN/AEI/ 10.13039/501100011033 and by "ERDF A way of making Europe", by the "European Union" or by the "European Union NextGenerationEU/PRTR", by European Commission under IBERIFIER Plus - Iberian Digital Media Observatory (DIGITAL-2023-DEPLOY-04-EDMO-HUBS, Project Nº: 101158511), by Calouste Gulbenkian Foundation, under the project MuseAI - Detecting and matching suspicious claims with AI and by "Convenio Plurianual with the Universidad Politécnica de Madrid in the actuation line of Programa de Excelencia para el Profesorado Universitario"

using the "Solar Power Dataset (4 Seconds Observations)" from the Monash benchmark [2] (resampled from 20 seconds to 10 minutes), and shows the outcomes. The rest of the paper is organized as follows. Section 2: DeepVATS description, provides a basic description of the visual application. Section 3: Scalability Analysis, examines the most important steps within the execution.

2 DeepVATS description

DeepVATS integrates three main modules: Deep Learning Module, Storage Module and Visual Analytics Module. The Deep Learning Module is a python library developed using Jupyter Notebooks and nbdev [3]. It facilitates datasets management with Weights & Biases (W&B, wandb.ai) and neural network fastai learners training [4]. There are two algorithms for data encoding implemented: Deep Convolutional auto-encoder (DCAE) [1] and Masked Value Prediction (MVP) [5]. The Storage module refers to W&B as a "database" for saving the dataset and encoder artifacts. The Visual Analytics module, the interactive component of DeepVATS, uses TSAI's *get_acts_and_grads* [6] to get model's embeddings and project them in 2D using UMAP [7], TSNE or PCA (rapidsai's *cuml* implementations [8]). Then, *hdbscan* is used for clustering [9] and the interactive plots are shown. These plots include the embeddings' projection plot (PP) and the time series plot (TSP). By selecting points in the PP, their corresponding windows are displayed in the TSP and vice versa. To enhance performance, the number of timepoints to display in the TSP is limited to 10K, showing the position of windows below the TSP. Additional features like zoom functionality aid in the detailed analysis and better understanding of the embeddings. DeepVATS presents three key advantages over its competitors (e.g. TimeCluster [1]). First, its MVP backbone model enables versatile use beyond segmentation. Second, the model is trained across different windows sizes, reducing sensitivity to its choice. Last, its modular open-source design facilitates easy adaptation to new tasks, such as outliers detection [5].

3 Scalability analysis

To check the app scalability and ensure its performance in large time series, DeepVATS is analysed using the Monash benchmark [2]. It offers more than 20 datasets varying the number of series and their lengths. The largest dataset is the Solar Power Dataset (4 Seconds Observations). It contains a single time serie with the solar power production recorded with a 4s frequency along one year, getting up to 7.5M elements. This dataset has been resampled into smaller ones based in a frequency factor. As 10 minutes frequency has previously been used in Monash benchmark, factors 5, 8, 15, 32, 75 and 150 have been selected, getting frequencies from 8s to 10m, resulting from 3.7M to 49.3K elements. The following steps are executed and analysed for each subset. First, load the dataset from the feather file so the app automatically gets the embeddings from its last associated encoder and applies DR via UMAP using GPU and generates the associated PP and TSP. Second, change to PCA for its special interest, as detailed in the next sections. Third, use CPU. Fourth, clustering. Fifth, projection points selection. Sixth, plots interactions: PP zoom in and zoom out, and point selection on the TSP. Finally, PP aesthetics updates: update point alpha and removing lines in PP. The next sections summarize the step's logs analysis outcomes.

3.1 Load dataset and get embeddings

The TSP is generated using *dygraph* [10] function, which requires a data frame with timestamps as row names. Converting the time index column to row names significantly adds execution time in large time series. Instead, using $xts(tsdf, order.by = tsdf\$timeindex)$ direct conversion before *dyhraph* call has significantly reduced the execution time (down to 27 seconds for the 4s dataset).
In the get embeddings step, time and memory consumption rapidly increases when adding points to the dataset. For better memory handling and reduced communications, *get_enc_embs* function has been modified to stride encoder input directly over the python object and get embeddings by chunks.

3.2 Compute projections

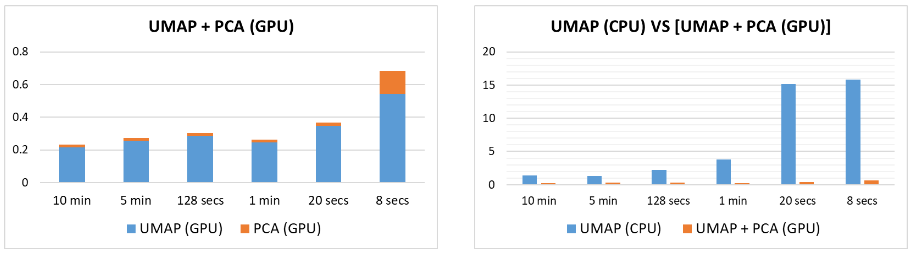

Fig. 1. Compute projections execution times (seconds). By the left UMAP + PCA (GPU) aggregated plot. By the right, the comparison with UMAP (CPU).

The computation of projections is known to be much faster in GPU than CPU. However, there is a bug in *cuml* that makes UMAP function fail, resulting on unstable projections and low quality clusters [11], [12]. To avoid this problem, CPU implementation can be used, but this results on larger execution time. Thus, there are two possible solutions: change the GPU UMAP implementation (e.g., use Peter Eisenmann's parallel UMAP implementation [13]); or explore the execution of PCA followed by UMAP as proposes Corey J. Nolet in order to obtain better results. This is not implemented, but as **Fig. 1** shows, the time would be much better than using CPU.

3.3 Compute and visualize plots

A reduced performance was detected when loading solar 4 seconds dataset. Thus, the scalability analysis was started and we saw that reactive operations are being recalculated. This adds a lot of time to the rendering of the plots. Also, a notably difference between execution and rendering time was noticed, getting near to 4 minutes to render plots for 4 seconds frequency dataset. To avoid recalculations, cache can be used via *reactiveVal* and cache plots [14] [15].

3.4 Compute clusters

The clusters computation is really fast even for large time series. However, some reactive functions have been detected that affects to the performance of this part of the execution. Thus, the cache analysis will also be useful for enhancing this step.

4 Conclusions and future work

DeepVATS is a powerful tool that can be used for visually analyse time series in an easy way for both univariate and multivariate time series. However, some scalability aspects must be enhanced for it to be usable for large time series. Thus, the next steps are: improving the use of cache in the shiny app for ensuring no recalculations; and explore PCA followed by UMAP execution and other UMAP and other UMAP implementations for ensuring good GPU usability.

References

1. M. Ali, M. W. Jones, X. Xie, and M. Williams, 'TimeCluster: dimension reduction applied to temporal data for visual analytics', *Vis. Comput.*, vol. 35, no. 6, pp. 1013–1026, Jun. 2019, doi: 10.1007/s00371-019-01673-y.
2. R. Godahewa, C. Bergmeir, G. I. Webb, R. J. Hyndman, and P. Montero-Manso, 'Monash Time Series Forecasting Archive'.
3. 'nbdev – Create delightful software with Jupyter Notebooks'. Accessed: Jan. 04, 2024. [Online]. Available: https://nbdev.fast.ai/
4. 'fastai/fastai'. fast.ai, Jan. 04, 2024. Accessed: Jan. 04, 2024. [Online]. Available: https://github.com/fastai/fastai
5. V. Rodriguez-Fernandez, D. Montalvo, F. Piccialli, G. J. Nalepa, and D. Camacho, 'DeepVATS: Deep Visual Analytics for Time Series'. 2023.
6. 'tsai - Explainability'. Accessed: Jan. 04, 2024. [Online]. Available: https://timeseriesai.github.io/tsai/models.explainability.html#get_acts_and_grads
7. L. McInnes, J. Healy, and J. Melville, 'UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction'. arXiv, Sep. 17, 2020. doi: 10.48550/arXiv.1802.03426.
8. 'API Reference — cuml 23.12.00 documentation'. Accessed: Jan. 04, 2024. [Online]. Available: https://docs.rapids.ai/api/cuml/stable/api/#
9. 'scikit-learn-contrib/hdbscan'. scikit-learn-contrib, Jan. 04, 2024. Accessed: Jan. 04, 2024. [Online]. Available: https://github.com/scikit-learn-contrib/hdbscan
10. 'Using in Shiny Applications'. Accessed: Jan. 04, 2024. [Online]. Available: https://rstudio.github.io/dygraphs/shiny.html
11. '[BUG] Accuracy of lanczos solver. · Issue #313 · rapidsai/raft', GitHub. Accessed: Jan. 04, 2024. [Online]. Available: https://github.com/rapidsai/raft/issues/313

12. ‘[BUG] different outputs for UMAP on CPU vs. GPU · Issue #5474 · rapidsai/cuml’, GitHub. Accessed: Jan. 04, 2024. [Online]. Available: https://github.com/rapidsai/cuml/issues/5474
13. Peter, ‘p3732/gpumap’. Oct. 16, 2023. Accessed: Jan. 04, 2024. [Online]. Available: https://github.com/p3732/gpumap
14. ‘reactiveVal’. Accessed: Jan. 04, 2024. [Online]. Available: https://shiny.posit.co/r/reference/shiny/1.1.0/reactiveval
15. ‘renderCachedPlot’. Accessed: Jan. 04, 2024. [Online]. Available: https://shiny.posit.co/r/reference/shiny/1.3.0/rendercachedplot