SELECTING LANGUAGE MODELS FEATURES VIA SOFTWARE-HARDWARE CO-DESIGN

Vlad Pandelea[⋆] *Edoardo Ragusa*[†] *Paolo Gastaldo*[†] *Erik Cambria* [‡]

[⋆] Continental-NTU Corporate Lab, Nanyang Technological University, Singapore
[†] University of Genoa, Italy, [‡] Nanyang Technological University, Singapore

ABSTRACT

The availability of new datasets and deep learning techniques have led to a surge of effort directed towards the creation of new models that can exploit the large amount of data. However, little attention has been given to the development of models that are not only accurate, but also suitable for user-specific use or geared towards resource-constrained devices. Fine-tuning deep models on edge devices is impractical and, often, user customization stands on the sub-optimal feature-extractor/classifier paradigm. Here, we propose a method to fully utilize the intermediate outputs of the popular large pre-trained models in natural language processing when used as frozen feature extractors, and further close the gap between their fine-tuning and more computationally efficient solutions. We reach this goal exploiting the concept of software-hardware co-design and propose a methodical procedure, inspired by Neural Architecture Search, to select the most desirable model taking into consideration application constraints.

Index Terms— Evaluation Methodologies, Opinion Mining / Sentiment Analysis, Language Models, Edge Computing

1. INTRODUCTION

The increase in computational capacity, and the availability of large amounts of data, have led to a proliferation of deep learning models capable of exploiting these resources. Consequently, a lot of research has focused on developing novel deep learning models that aim to improve generalization performance in fields such as natural language processing (NLP) and sentiment analysis [1]. While accurate, these models stand on heavy computations, non-convex training procedures, and require huge memories. These side effects limit the development of user specific models, and the deployment on embedded devices. On the other hand, these are two major limitations if one thinks that smart devices continuously sample users' data and could produce better services by including user-specific inference functions. On these devices it is reasonable to train more shallow classifiers which, however, means losing the most important capability of end-to-end training for NLP.

This study is supported under the RIE2020 Industry Alignment Fund – Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contribution from the industry partner(s).

General purpose fixed representation combined with classifiers are largely sub-optimal compared to fine tuned deep models, due to the fact that the intermediate representations are also adjusted during the optimization process. To reduce this gap we design a strategy to fully take advantage of the powerful feature extraction capabilities of large pre-trained models that can be used for inference, but not training due to the prohibitive cost of the latter. The method aims to give the classifier access to multiple layers of the deep neural network. The rationale is as follows: fine tuning performs a transformation f of the original representation of the layers. Here, the same task is left to the classifier that in the ideal case will find the correct transformation for the different feature sets. Note that we do not seek for the optimal transformation f that would require hidden layers co-optimization. We rely on a sub-optimal representation that accesses information of multiple layers simultaneously.

Different layers of deep networks contain different semantic information [2]. The key point in the proposed approach is retrieving the right set of connections, which impacts the computational cost and the generalization capabilities. To solve this, we propose an architecture search inspired by Neural Architecture Search (NAS) [3] methods. The search is driven by the concept of software-hardware co-design [4] at application level, that is new in sentiment analysis. This task is of particular interest for multiple reasons. First, sentiment analysis state-of-the-art solutions are known to rely on large pre-trained models, which are fine-tuned to maximize performance [5], but strategies on how to fully utilize their feature extraction capabilities without fine-tuning are understudied. Moreover, sentiment is user-specific to a great extent and thus being able to deploy customized solutions is an important but overlooked aspect.

To the best of our knowledge, this is the first work that introduces the concept of software-hardware co-design for sentiment analysis. As a unique contribution, we present a new cost function that yields the selection of the final model taking into consideration not only the generalization performances but also the constraints of the application, i.e. latency and hardware resources of the embedded system. We perform a design space exploration using the feature extraction/classification paradigm, and focus on the classification stage.

During the design stage, designers can use the proposed design space exploration strategy to select the optimal combination of algorithms and hardware for the specific application among a set of candidates. Later, the selected training strategy can be deployed on the hardware and allow training on novel data directly on the device. The contributions of the paper can be summarized as follows: 1) A methodology to fully utilize the feature extraction capabilities of large pre-trained NLP models, at little cost following deployment; 2) A novel design strategy that allows to include hardware constraints directly in the learning phase, which, once the final model is chosen, supports on-device training; 3) A framework that trades-off application constraints satisfaction and generalization performance; the framework is compared against baseline solutions and proves more effective in balancing computational cost in two popular datasets for sentiment analysis in conversation.

2. RELATED WORK

The deployment of training phase of inexpensive, resource-constrained devices is an open issue, where random based neural network can be regarded as a lower bound in terms of computational cost [6]. Solutions deployed on reconfigurable platforms such as FPGAs [7, 8], may prove fast, yet expensive. Along the proposed architectures, these works introduced efficient strategies for hidden weights generation that reduced memory requirements dramatically. By contrast, implementations on micro-controllers or micro-computers have drawn limited attention, in spite of the fact that these devices best fit IoT applications and remarkably shrink the time-to-market of commercial products [9]. Recent approaches [10] combined ensemble mechanisms with random based networks. Biologically-inspired optimization stimulated self-adaptive evolutionary single layer networks [11], but proved computationally demanding.Limited work has explored the trade-off between computational cost and generalization performance for sentiment analysis. [12] demonstrated that hardware-friendly classifiers can achieve competitive performance on edge devices, [13] applied ELM to features extracted from speech data for emotion recognition, [14] used an ensemble of GPU and ELM classifiers for multimodal sentiment analysis.

3. APPROACH

Our framework allows training directly on embedded devices. We propose a NAS-like strategy to draw the most benefit from a frozen pre-trained language model by devising an algorithm that can search through its intermediate representations the ones that are most suitable for the task. To guide this search, as well as the selection of a suitable classifier, we introduce a cost function whose solution yields the most suitable model and set of connections taking into account performance, training time and training memory requirements.

3.1. Searching for the optimal connections

Typically, a hidden state of the last layer of language models is used for downstream tasks. However, because our feature extractor is not fine-tuned, it is sensible to use the prior information within the intermediate layers as potential features instead. This remains sub-optimal compared to a fine-tuning of the whole feature extractor, as that would allow for representation co-optimization. It does, however, allow for the classifier placed on top of the feature extractor to perform a transformation of multiple representations, simultaneously, which is expected to contain additional information that would not be present in, for instance, the last hidden state alone.

To select a combination of the most suitable features, we adopt a genetic algorithm for feature selection whereby a random set of features is selected as the candidate in the initialization step. Afterwards, a series of clones of the candidate are created and to each clone a mutation is applied, according to a pool of candidate operations and available feature sets (feature extractor intermediate outputs). In particular, to each of the clones we apply one of the following operations: 1) **ADD**, which concatenates a features vector selected at random to the current feature set; 2) **SUBSTITUTE**, which replaces a vector in the current set; 3) **DELETE**, which deletes a vector from the current set. Then, each resulting mutation is evaluated according to a performance measure. The best performing mutation is selected as the winner and retained for the next iteration. New candidate mutations are then created from it, starting a new cycle.

3.2. A hardware-aware cost function

Within the limited resources environment, we need an adequate measure of the performance of different representations, as well as different possible models. To address these concerns and propose a practical method to select a suitable model and intermediate representations from the feature extractor, we introduce a minimization objective over a pool of candidate solutions that yields a solution aware of the constraints of the environment that the final model is to be deployed in. Given a set of models, we want to select the model i^* that minimizes the performance error cost $c_{pi} = 1 - p_i, p_i \in [0, 1]$, where p_i can be any metric deemed to best represent the discrepancy between the predictions and the real labels. This however does not take into account hardware resources that are necessary to implement the models, which may be a limiting factor in real applications. Moreover, the solution of this problem is subject to hard constraints dictated by the maximum memory cost, M, the maximum training time cost to be tolerated, S, and the minimum performance to be achieved, denoted by the maximum cost in performance, P. This results in the following optimization problem, or cost function:

$$i^* = argmin_i \lambda_p c_{pi} + \lambda_m c_{mi} + \lambda_s c_{si} \quad (1)$$

$$0 < c_{mi} \leq M, 0 < c_{si} \leq S, 0 \leq c_{pi} \leq P \quad (2)$$

where c_{mi}, c_{si} are the memory and training time costs normalized normalized in $\in [0, 1]$. $\lambda_p, \lambda_m, \lambda_s$ are weights.

Because the application of the genetic algorithm for the selection of connections between the feature extractor and the classifier does not depend on the derivability of the function, we can use any cost function and set of constraints. In fact, the proposed cost function is quite general and admits a large set of options to estimate c_{mi} and c_{si}. Additionally, the selection of the relative weights $\lambda_p, \lambda_m, \lambda_s$ of each component is application specific. They could be, for instance, adjusted based on user feedback. As a result, we cannot make assumptions on the optimization procedure. However, we can use this cost function to select the optimal solution only among the set of candidates models available. Since the training of the models to evaluate their performance is orders of magnitude more computationally costly than evaluating the value of the cost function, the limiting factor tends to be the practical estimation of the components of the minimization problem.

3.3. Pipeline

In the previous sections we introduced a cost function that facilitates the choice of a model and set of connections suitable for the constrained target environment. Now, we present the pipeline that can be deployed in this environment. In theory, finding the optimal pipeline given hardware constraints is an open problem approached via NAS, which attempts to optimize the entire end-to-end pipeline at once, but may be too costly to solve. We then restrict the hypothesis space by fixing a-priori role and structure of our models, as seen in Fig. 1.

The building blocks identify the **Features Extraction** step, in which we have the feature extractor from which we select the most appropriate connections through the algorithm in Section 3.1, followed by **Dimensionality Reduction**, and the **Classification Stage**. Accuracy and the computational cost are determined by all the modules. Thus, one should design each module to trade-off these two quantities. The absolute optimal procedure should always consider all these steps. However, some design choices can be made a-priori, simplifying the search space, i.e. reducing the number of candidate solutions.

In particular, as a feature extractor we employ MobileBERT [15], a lightweight language model inspired to BERT-large. We do not fine-tune this feature extractor, as it would result in a prohibitive model selection cost. We further reduce the dimensionality of the features through Principal Component Analysis (PCA), for which we determine the number of dimensions via the Kneedle algorithm [16]. PCA, once trained, reduces the subsequent computations at the expense of a single matrix multiplication.

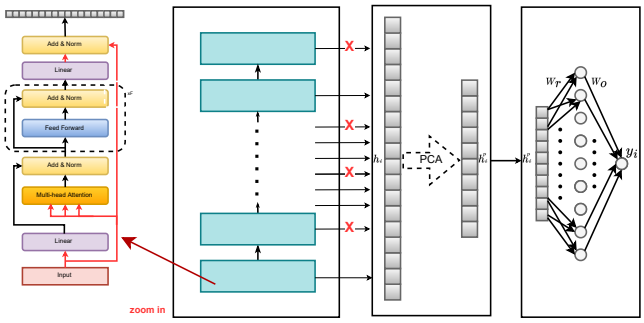

Fig. 1. High-level processing pipeline.

Moreover, since we employ multiple representations from different layers, redundancy in the data that can be effectively reduced via PCA. Finally, the classification stage is the only trainable one in our framework. We consider a search space limited to single layer feed forward neural networks (SLFNs), by virtue of the excellent trade-off between generalization performances and computational cost that they present [6]. To train these, we consider the following paradigms: 1) **Linear**, equivalent to fine tuning only the last layer of the feature extractor; 2) **Hidden**, in which an activation is placed on a hidden layer; 3/4) **Lasso/Ridge**, denoting random based solutions. In particular, we employ ELM [17]. These solutions differ considerably in memory and time consumption. Accordingly, they are expected to offer an optimal solution based on the requirements of the application scenario.

4. EXPERIMENTAL RESULTS

We perform the experiments on the two datasets, CMU-MOSEI [18] and MELD [19]. CMU-MOSEI contains around 22000 utterances, while MELD contains around 13000. We study two cases based on the dataset size: all the available training samples (full dataset), and 1000 samples (small dataset). The second case is of interest as it represents a more realistic scenario, for user-tailored applications on resource-constrained devices. The baseline in our system is MobileBert [15], hereinafter **MobBert**, fine-tuned on the dataset. For the first set of experiments we only use the output of the last layer as typically done, whereas later we will show the benefit of a feature selection strategy considering the intermediate layers. In line with previous work we use *Weighted Accuracy* as the metric for CMU-MOSEI [20] and *Weighted F1 score* for MELD [21]. We empirically measure time and memory requirements of the training phase of the models on an Intel(R) Xeon(R) W-2235 CPU @ 3.80GHz, using wall-clock time and peak memory usage. The system will allocate an oversized quantity of memory if necessary, which will ensure that on a embedded system the memory required will be lower or equal than the one measured on such device. Similarly, the measured latency will be a lower bound.

However, the relative placement of different algorithms will help in selecting the correct device. In addition, performing the measures inside the candidate embedded systems would bring in additional steps to adapt the code for every single device, leading to increased time to market and costs.

4.1. Performance and computational trade-off

Our experiments do not aim to prove that the proposed solution yields state-of-the-art generalization performance. It is well-known that deep learning frameworks are more accurate. Instead, we show that our strategy can effectively incorporate hardware constraints. We investigate which models are selected as the importance given to the performance metric, memory, and training time requirements vary, by performing a parametric study over the values of the normalized λ_p, λ_m and λ_s. For each configuration we compute the average over the 100 trials, then for each value of $(\lambda_p, \lambda_m, \lambda_s)$ we show the type of the best performing model, for the two cases of different dataset size for CMU-MOSEI, in Fig. 2. On the axes we have the weights given to λ_p, λ_m, and λ_s. Each point in the resulting 3D scatter plot denotes the type of model minimizing the cost function for the given parameters.

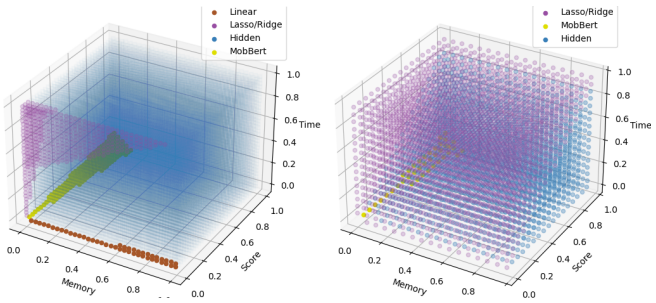

Fig. 2. Plot of the class of models minimizing the cost function on CMU-MOSEI. Full dataset (left), small dataset (right).

We find that, regardless of dataset size, heavy transformer based models such as **MobBert** only win the trade-off when extreme importance is given to the performance and little to none to memory and training time. When the dataset is large, solutions based on **Lasso/Ridge** may be memory intensive as they may involve matrix inversion. However they are generally still faster than backpropagation as seen from the figure, observing that, when high importance is given to time, these solutions are preferred. In most cases, however, a neural network with a hidden layer proves to be the most effective. When the dataset size is smaller, the **Lasso/Ridge** based models are chosen more frequently, since their memory and time requirements grow superlinearly with respect to data size. Thus, these models can be very effective for user-tailored applications on embedded devices, where they can be quickly fine-tuned without exceeding the device's capabilities. Similar trends were observed for the MELD dataset.

	CMU-MOSEI		**MELD**	
	WA	*#Feats*	*F1*	*#Feats*
Ridge - last layer	0.7762	127	0.5808	119
MobBERT	0.8398	N.A.	0.6530	N.A.
Ridge - genetic	0.8074	290	0.6192	265

Table 1. Comparison of the different methods.

4.2. Feature selection via genetic algorithm

In order to investigate whether the gap between models were the pre-trained feature extractor is frozen and those in which it is fine-tuned can be further closed, we employ a genetic algorithm for feature selection that leverages the intermediate outputs of our feature extractor, as detailed in Section 3.1. We do not repeat the full analysis including time and memory requirements, which can be easily incorporated by setting $\lambda_p \neq 0$ and $\lambda_s \neq 0$. We employ the **Ridge** solution as the model choice. We compare the performance on the same data split of this strategy with two baselines, the **Ridge** solution and **MobBert**. We perform 3 trials, each with a different initialization of the feature set of the genetic algorithm. For simplicity, we limit the number of intermediate outputs that can be concatenated to a maximum of 5, the number of mutations is set to 5, and the number of iterations is set to 200.

Results are reported in Table 1. **Ridge - last layer** denotes the **Ridge** model utilizing the output of the last layer of the transformer model, **MobBERT** the fine-tuned transformer and **Ridge - genetic** the **Ridge** model whose input features are selected via the genetic algorithm. We find that adding the feature selection step during the initial model selection leads to a noticeable improvement in performance, over 3% on CMU-MOSEI and nearly 4% for MELD with respect to the typical method of using the output of the last layer of the feature extractor. The number of input features also increases but this leads to a relatively low increase in computational requirements as the main computational load is due to the model's internal architecture, which remains untouched.

5. CONCLUSION

In this work we have shown how to make use of the feature extraction capabilities of large pre-trained language models for NLP to deploy solutions targeted for resource constrained devices, paving the way for user-specific models. We did so by presenting a new approach to model selection that is better suited for real world applications. Shifting the focus from the predominant mindset of developing increasingly complex models aimed at improving performance, our experiments demonstrated that a framework including standard classifiers such as SLFNs can outperform state-of-the-art solutions when considering realistic constraints, and pave the way for the development of user-specific inference functions.

6. REFERENCES

[1] Jingfeng Cui, Zhaoxia Wang, Seng-Beng Ho, and Erik Cambria, "Survey on sentiment analysis: Evolution of research methods and topics," *Artificial Intelligence Review*, vol. 56, 2023.

[2] Yuan Huang, Zhixing Li, Wei Deng, Guoyin Wang, and Zhimin Lin, "D-bert: Incorporating dependency-based attention into bert for relation extraction," *CAAI Transactions on Intelligence Technology*, vol. 6, no. 4, pp. 417–425, 2021.

[3] Jaeseong Lee, Jungsub Rhim, Duseok Kang, and Soonhoi Ha, "S3nas: Fast hardware-aware neural architecture search methodology," *IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[4] Christian Gianoglio, Edoardo Ragusa, Paolo Gastaldo, and Maurizio Valle, "A novel learning strategy for the trade-off between accuracy and computational cost: a touch modalities classification case study," *IEEE Sensors Journal*, 2021.

[5] Erik Cambria, Qian Liu, Sergio Decherchi, Frank Xing, , and Kenneth Kwok, "SenticNet 7: A commonsense-based neurosymbolic AI framework for explainable sentiment analysis," in *LREC*, 2022, pp. 3829–3839.

[6] Weipeng Cao, Xizhao Wang, Zhong Ming, and Jinzhu Gao, "A review on neural networks with random weights," *Neurocomputing*, vol. 275, pp. 278–287, 2018.

[7] Jose V Frances-Villora, Alfredo Rosado-Muñoz, Manuel Bataller-Mompean, Juan Barrios-Aviles, and Juan F Guerrero-Martinez, "Moving learning machine towards fast real-time applications: A high-speed fpga-based implementation of the os-elm training algorithm," *Electronics*, vol. 7, no. 11, pp. 308, 2018.

[8] Amin Safaei, QM Jonathan Wu, Thangarajah Akilan, and Yimin Yang, "System-on-a-chip (soc)-based hardware acceleration for an online sequential extreme learning machine (os-elm)," *IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[9] Peter Alaba, Segun Popoola, Lanre Olatomiwa, Mathew Akanle, Olayinka Ohunakin, Emmanuel Adetiba, Alex Opeoluwa, Aderemi Atayero, and Wan Daud, "Towards a more efficient and cost-sensitive extreme learning machine: A state-of-the-art review of recent trend," *Neurocomputing*, vol. 350, pp. 70–90, 2019.

[10] Edoardo Ragusa, Christian Gianoglio, Rodolfo Zunino, and Paolo Gastaldo, "Random-based networks with dropout for embedded systems," *Neural Computing and Applications*, pp. 1–16, 2020.

[11] Tianqi Wu, Min Yao, and Jianhua Yang, "Dolphin swarm extreme learning machine," *Cognitive Computation*, vol. 9, no. 2, pp. 275–284, 2017.

[12] Vlad Pandelea, Edoardo Ragusa, Tommaso Apicella, Paolo Gastaldo, and Erik Cambria, "Emotion recognition on edge devices: Training and deployment," *Sensors*, vol. 21, no. 13, pp. 4496, 2021.

[13] Kun Han, Dong Yu, and Ivan Tashev, "Speech emotion recognition using deep neural network and extreme learning machine," in *Fifteenth annual conference of the international speech communication association*, 2014.

[14] Ha-Nguyen Tran and Erik Cambria, "Ensemble application of elm and gpu for real-time multimodal sentiment analysis," *Memetic Computing*, vol. 10, no. 1, pp. 3–13, 2018.

[15] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou, "Mobilebert: Task-agnostic compression of bert for resource limited devices," *ICLR Openreview*, 2020.

[16] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan, "Finding a" kneedle" in a haystack: Detecting knee points in system behavior," in *2011 31st international conference on distributed computing systems workshops*. IEEE, 2011, pp. 166–171.

[17] Guang-Bin Huang, Lei Chen, Chee Kheong Siew, et al., "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Trans. Neural Networks*, vol. 17, no. 4, pp. 879–892, 2006.

[18] AmirAli Bagher Zadeh, Paul Pu Liang, Soujanya Poria, Erik Cambria, and Louis-Philippe Morency, "Multimodal language analysis in the wild: Cmu-mosei dataset and interpretable dynamic fusion graph," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 2236–2246.

[19] Soujanya Poria, Devamanyu Hazarika, Navonil Majumder, Gautam Naik, Erik Cambria, and Rada Mihalcea, "MELD: A multimodal multi-party dataset for emotion recognition in conversations," in *ACL*, 2019, pp. 527–536.

[20] Aman Shenoy and Ashish Sardana, "Multilogue-net: A context aware rnn for multi-modal emotion detection and sentiment analysis in conversation," *arXiv preprint arXiv:2002.08267*, 2020.

[21] Yazhou Zhang, Qiuchi Li, Dawei Song, Peng Zhang, and Panpan Wang, "Quantum-inspired interactive networks for conversational sentiment analysis.," 2019.