LEARNING TO RANK: A PROGRESSIVE NEURAL NETWORK LEARNING APPROACH

Dat Thanh Tran[1] *and Alexandros Iosifidis*[2]

[1]Laboratory of Signal Processing, Tampere University of Technology, Finland
[2]Department of Engineering, ECE, Aarhus University, Denmark
Emails: dat.tranthanh@tut.fi, alexandros.iosifidis@eng.au.dk

ABSTRACT

Learning to rank is an essential component in an information retrieval system. The state-of-the-art ranking systems are often based on an ensemble of classifiers, such as Random Forest or LambdaMART, which aggregates the ranking outputs produced by thousands of classifiers. The storage and computation requirement of an ensemble model is usually very high, imposing a significant operating cost to the retrieval system. To tackle this problem, we propose an algorithm that adaptively learns a single heterogeneous feedforward network architecture, composing of Generalized Operational Perceptrons, given a ranking problem. Experimental results in web search ranking and image retrieval tasks show that the proposed algorithm compares favourably to the related algorithms.

Index Terms— Generalized Operational Perceptron, Progressive Neural Network Learning

1. INTRODUCTION

The Learning to Rank (LETOR) problem finds its application in several domains, such as online advertisements [1], recommendation systems [2], natural language processing [3] or multimedia search engines [4]. The goal of a LETOR problem is to learn a ranking model that produces a ranked list of data items given a query. Based on the input and the output of the underlying ranking method, models can be divided into three categories: those following a pointwise, a pairwise or a listwise approach. Since the problem of ranking inherently involves evaluating the relative relevance, the most popular methods are usually based on pairwise or listwise approach. In pairwise ranking, a model is learned to correctly assign a relevance score for each pair of query i and database item j, while in listwise ranking, a model is learned to produce the correct permutation of data items given a query.

Several state-of-the-art ranking systems employ an ensemble of regression trees such as LambdaMART [5], BROOF-L2R [6] or Quickscorer [7]. The advantage of forest-based models is the ease of parallelization since individual regressors can be trained in parallel. In addition, traditional information retrieval metrics can also be used as a loss function in tree-based models. Nevertheless, these models often consist of thousands of weak classifiers, imposing huge memory and computation cost in total, which is critical in a system dealing with millions of concurrent queries.

There has been a great effort to improve the efficiency of LETOR models recently [7]. For example, in [8], the authors proposed to approximate the scoring function of an ensemble by training a feed-forward network to mimic the output of a pre-trained ensemble. This approach is also known as knowledge distillation in the network compression community. Expert-designed neural network architectures have also been proposed in [9, 10]. In [7], the authors proposed an algorithm to efficiently traverse additive ensembles of regression trees by only using bitwise operations, producing significant ranking speedup.

In this paper, we propose to tackle the LETOR problem by adaptively growing a neural network architecture comprised of Generalized Operational Perceptrons (GOP), which is a generalized Perceptron model proposed in [11], to better simulate the neuronal activities of biological neurons. Different from the traditional McCulloch-Pitts perceptron model, GOP admits a wide range of both linear and nonlinear transformations. It has been shown in [11] that GOP-based architecture can surpass the learning capacity of Multilayer Perceptrons (MLPs) or Radial Basis Function (RBF) networks with fewer neurons. Thus, given a particular LETOR problem at hand, we aim to harness the capacity of GOPs to learn a compact and efficient network topology progressively.

The contributions of our work can be summarized as follows:

- We propose to solve the ranking problem by a data-dependent, efficient algorithm that progressively learns a heterogeneous multilayer architecture of GOPs. The compact network architectures generated by our proposed algorithm requires small memory and computation footprint during inference.
- We provide empirical validation of our algorithm on web search ranking and image retrieval tasks in comparison with related algorithms.

The remainder of the paper is organized as follows: Sec-

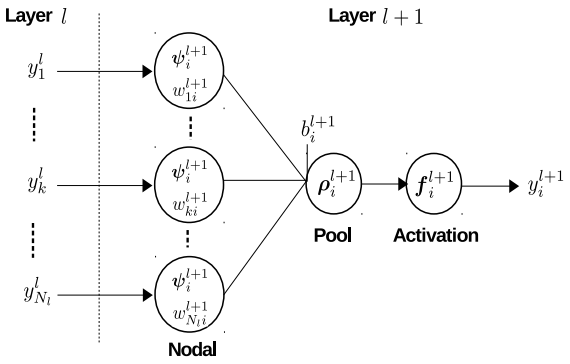

Fig. 1. Activities of the i-th GOP neuron at layer $l+1$, characterized by the synaptic weights w_{ki}^{l+1}, the nodal operator $\boldsymbol{\psi}_i^{l+1}$, the pooling operator $\boldsymbol{\rho}_i^{k+1}$ and the activation operator $\boldsymbol{f}_i^{l+1}$

tion 2 reviews the GOP model and its original learning algorithm. Section 3 describes our proposed algorithm. In Section 4, we detail our experimental setup and present our empirical results. Section 5 concludes our work.

2. RELATED WORKS

2.1. Generalized Operational Perceptron

A GOP neuron sequentially performs three distinct operations: nodal ($\boldsymbol{\psi}$), pooling ($\boldsymbol{\rho}$) and activation ($\boldsymbol{f}$). These three operations simulate the operations performed in biological learning system of mammels: the modification of the input signals in the Dendrites; the pooling of the modified signals in the Soma; and the pulse sending when the pooled potentials exceed a threshold in Axon hillock [11]. Since a diverse set of neuronal activities is performed in a biological learning system [12], GOP neuron model encapsulates this diversity of transformations by a set of nodal, pooling and activation operators, from which each GOP neuron can select. In this paper, the term *operator set* refers to a particular choice of nodal, pooling and activation operators. The operations of a GOP neuron is illustrated in Figure 1 and the library of operators used in our work is given in Table 1. The following equations describe the nodal, pooling and operation of the i-th GOP in layer $l+1$:

$$z_{ki}^{l+1} = \boldsymbol{\psi}_i^{l+1}(y_k^l, w_{ki}^{l+1}) \tag{1}$$

$$x_i^{l+1} = \boldsymbol{\rho}_i^{l+1}(z_{1i}^{l+1}, \ldots, z_{N_l i}^{l+1}) + b_i^{l+1} \tag{2}$$

$$y_i^{l+1} = \boldsymbol{f}_i^{l+1}(x_i^{l+1}) \tag{3}$$

Where w_{ki}^{l+1} and b_i^{l+1} denote the adjustable synaptic weights and bias term, respectively.

2.2. Progressive Operational Perceptron

It is clear that learning GOP-based networks involves finding the proper operator set and the weights for each neuron. To this end, the Progressive Operational Perceptron (POP) algorithm was proposed to train such networks [11]. Given a target Mean Squared Error (MSE) and a network template that defines the maximum number of hidden layers and the size of each layer (h_k), POP sequentially learns one hidden layer at a time with the assumption that GOPs in the same hidden layer share the same operator set. At step k, all previous $k-1$ hidden layers are fixed and POP constructs a Single Hidden Layer Network (SHLN) consists of h_{k-1} input neurons, h_k hidden GOPs and O output GOPs. The weights and the operator set of the hidden (ϕ_h) and output layer (ϕ_O) in SHLN are determined by a greedy iterative search procedure called two-pass GIS.

Let N be the number of operator sets in the library, and ϕ_h and ϕ_O be the operator set of the hidden and output layer of SHLN. In the first pass, ϕ_h is chosen randomly, and POP assigns each operator set in the library to ϕ_O and trains the SHLN configuration with E epochs using Back Propagation (BP) algorithm. The best ϕ_O^* providing the minimum MSE is then selected. With ϕ_O^* fixed, POP iterates through the library again to select the best ϕ_h^*. The second GIS pass is similar to the first pass, except ϕ_h^* is used instead of a random one. It is clear that the computation cost of two-pass GIS is NE BP epochs.

After two-pass GIS, if the target MSE is achieved, POP terminates. Otherwise, the algorithm continues to learn the $(k+1)$-th hidden layer in a similar manner. After the progression, POP fixes all operator set assignments and fine-tunes the weights for some epochs if the target MSE is not met.

3. PROPOSED ALGORITHM

While POP adopts a layer-wise approach with fixed layer sizes and layer homogeneity constraint, we propose an algorithm, called Heterogeneous Multilayer Generalized Operational Perceptron (HeMLGOP), which performs block-wise progression leading to heterogeneous layers of data-dependent size.

3.1. Network Progression

HeMLGOP operates on a block unit; thus the number of GOPs in a block unit is given as a hyper-parameter. At each progressive step, HeMLGOP adds a new block of GOPs to the existing network architecture, and all previous blocks are fixed (the weights and the operator set assignment). At step k, given that l hidden layers have been learned, there are two cases:

- If the progression in the l-th hidden layer was not terminated in step $k-1$, HeMLGOP adds new block to

Table 1. Operators

Nodal ($\mathbf{\Psi}$)	$\boldsymbol{\psi}_i^{l+1}(y_k^l, w_{ki}^{l+1})$
Multiplication	$w_{ki}^{l+1} y_k^l$
Exponential	$\exp(w_{ki}^{l+1} y_k^l) - 1$
Harmonic	$\sin(w_{ki}^{l+1} y_k^l)$
Quadratic	$w_{ki}^{l+1}(y_k^l)^2$
Gaussian	$w_{ki}^{l+1} \exp(-w_{ki}^{l+1}(y_k^l)^2)$
DoG	$w_{ki}^{l+1} y_k^l \exp(-w_{ki}^{l+1}(y_k^l)^2)$
Pool ($\mathbf{P}$)	$\boldsymbol{\rho}_i^{l+1}(z_{1i}^{l+1}, \ldots, z_{N_l i}^{l+1})$
Summation	$\sum_{k=1}^{N_l} z_{ki}^{l+1}$
1-Correlation	$\sum_{k=1}^{N_l-1} z_{ki}^{l+1} z_{(k+1)i}^{l+1}$
2-Correlation	$\sum_{k=1}^{N_l-2} z_{ki}^{l+1} z_{(k+1)i}^{l+1} z_{(k+2)i}^{l+1}$
Maximum	$\max_k(z_{ki}^{l+1})$
Activation ($\mathbf{F}$)	$\boldsymbol{f}_i^{l+1}(x_i^{l+1})$
Sigmoid	$1/(1+\exp(-x_i^{l+1}))$
Tanh	$\sinh(x_i^{l+1})/\cosh(x_i^{l+1})$
ReLU	$\max(0, x_i^{l+1})$
Softplus	$\log(1+\exp(-x_i^{l+1}))$
Inverse Absolute	$x_i^{l+1}/(1+\lvert x_i^{l+1}\rvert)$
ELU	$x_i^{l+1}\mathbf{1}_{x_i^{l+1}\geq 0} + \exp(x^{l+1})_i \mathbf{1}_{x_i^{l+1}<0}$

the l-th hidden layer.

- If the progression in the l-th hidden layer was terminated in step $k-1$, HeMLGOP forms the $(l+1)$-th hidden layer with a new block.

The progression in a hidden layer is terminated when the addition of new neurons does not improve the performance significantly. This saturation is determined based on the rate of performance improvement. Particularly, let $\mathcal{L}_{k-1}$ and $\mathcal{L}_{k-2}$ be the loss achieved at step $k-1$ and $k-2$ respectively, the progression in the current hidden layer terminates if:

$$\frac{\mathcal{L}_{k-2} - \mathcal{L}_{k-1}}{\mathcal{L}_{k-2}} < \epsilon \tag{4}$$

where ϵ is a hyper-parameter of the proposed algorithm.

HeMLGOP terminates of progression of the network when adding a new fully grown hidden layer merely improves the performance. This means that when the progression in the l-th hidden layer terminates, HeMLGOP evaluates the relative improvement of the network with and without the l-th hidden layer. That is, if:

$$\frac{\mathcal{L}_m - \mathcal{L}_n}{\mathcal{L}_m} < \epsilon \tag{5}$$

HeMLGOP discards the l-th hidden layer, leading to the final network architecture having $l-1$ hidden layers. Otherwise, the algorithm keeps the l-th hidden layer and continues growing new hidden layers. In Eq. (5), m and n refers to the last block index of hidden layer $l-1$ and l respectively.

3.2. Block optimization

In HeMLGOP, we enforce two constraints to allow efficient optimization of the new block: GOPs within the same block share the same operator set, and the output layer which connects the last hidden layer and the output neurons is a linear transformation layer. At step k, when a new block of GOPs is added to the network, HeMLGOP optimizes for the operator set assignment and the weights of the new block as well as the output layer weights.

To select the best performing operator set for the new block, it is necessary that all operator sets in the library are evaluated. We propose to perform the evaluation based on a randomized approach: for each operator set in the library, we randomly draw the block weights from a uniform distribution and the output layer weights are calculated via the least-squared solution of regressing the targets. Particularly, let $\mathbf{H} \in \mathbb{R}^{N \times D}$ and $\mathbf{Y} \in \mathbb{R}^{N \times C}$ denote the last hidden layer output and the target output. The output layer weights $\mathbf{W}_o$ are calculated as follows:

$$\mathbf{W}_o = \mathbf{H}^{\dagger}\mathbf{Y} \tag{6}$$

where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse of $\mathbf{H}$.

The best performing operator set with random weights is selected for the new block and fixed. Since each operator set represents a distinct type of transformation, we assume that the suitable functional form of a GOP, i.e., the operator set, can be evaluated with random synaptic weights. Even if a well suitable operator set is not selected in this step, it can always be selected when the next blocks are added to the current hidden layer.

After fixing the operator set assignment of the new block, HeMLGOP updates the block weights and output layer weights by BP for E epochs to harness the new block, which also helps avoid redundancy of "weak neurons". Once a block is optimized, its operator set and weights are fixed. At each progressive step, HeMLGOP only requires one loop through the library of operator sets, compared to four loops in POP, which is much more computationally expensive.

4. EXPERIMENTS

To evaluate the effectiveness of HeMLGOP in LETOR problems, we conducted experiments on two datasets:

- MSLR-WEB10K [13]: Microsoft Learning to Rank dataset contains 10000 queries. Each query-url pair is represented by a 136-dimensional feature vector and a relevance score ranging from 0 (irrelevant) to 4 (perfectly relevant). We used a subset of MSLR-10K by randomly selecting 500 query IDs (non-overlapping) for each train/validation/test set with 30 relevant and 30 irrelevant query-url pairs for ID. This results in $30,000$ pairs for each set.

Table 2. Performance on MSLR-WEB10K

	mAP	NDCG@1	NDCG@5	NDCG@10
SVM [16]	0.5718	0.3036	0.4435	0.5621
RF [17]	0.6117	0.5956	0.6851	0.7475
BLS [18]	0.5909	0.5614	0.6550	0.7270
S-ELM [20]	0.6070	0.5908	0.6794	0.7444
PLN [19]	0.5946	0.5304	0.6536	0.7248
HeMLGOP	**0.6143**	**0.5961**	**0.6884**	**0.7496**

Table 3. Performance on ImageNet

	mAP	NDCG@1	NDCG@5	NDCG@10
SVM [16]	0.5554	0.5950	0.5883	0.5928
RF [17]	0.7447	**0.9250**	0.8754	0.8439
BLS [18]	0.7169	0.8200	0.8191	0.8002
S-ELM [20]	0.7235	0.8750	0.8480	0.8298
PLN [19]	0.5666	0.8350	0.7723	0.7388
HeMLGOP	**0.8131**	0.9150	**0.9082**	**0.8910**

Table 4. Operating cost on MSLR-WEB10K

	Storage (Mb)	Inference (ms)
SVM [16]	6×10^{-3}	8.3×10^{-4}
RF [17]	4.9×10^{3}	4.7
BLS [18]	1.1	0.1
S-ELM [20]	33	3.7
PLN [19]	0.78	1.1×10^{-2}
HeMLGOP	0.06	4.7×10^{-1}

Table 5. Operating cost on ImageNet

	Storage (Mb)	Inference (ms)
SVM [16]	7×10^{-3}	1.4×10^{-2}
RF [17]	3.2×10^{3}	5.0
BLS [18]	1.8	4.7×10^{-1}
S-ELM [20]	22	2.1
PLN [19]	1.0	3.4×10^{-2}
HeMLGOP	0.1	5.6×10^{-1}

- Image Retrieval: ImageNet dataset [14] contains images representing different objects. We used a subset of this dataset to prepare the image retrieval task by selecting 200 different objects (non-overlapping) for each train/validation/test set. For each object, we randomly generated 50 relevant and 50 irrelevant query-image pairs, producing $20,000$ query-image pairs for each set. To generate features for each query-image pair, we took the difference between the deep features extracted by average pooling over spatial dimensions of the last convolution layer of VGG network [15] trained on ILSVRC2012.

Given the representation of query pairs with labeled relevance, the LETOR problem is posed as a classification problem, i.e., to learn the prediction of relevance score. Along with HeMLGOP, we evaluated linear SVM [16], Random Forest (RF) [17], BLS [18], PLN [19] and S-ELM [20]. The last three algorithms also perform progressive learning of the network architecture. Since POP is computationally expensive for large datasets, we did not conduct experiments with POP. For the full setting of hyper-parameters of each model in our experiments, please refer `https://github.com/viebboy/LETOR`. Regarding the performance metrics, mean Average Precision (mAP) and Normalized Discounted Cumulative Gain (NDCG) over each query ID on the test set are reported.

Table 2 and 3 show the experimental results on two datasets. For both datasets, it is clear that HeMLGOP outperforms other ranking algorithms. On MSLR-WEB10K dataset, the performance of Random Forest (RF) is on par with HeMLGOP while other competing algorithms lag behind, especially when the cut-off thresholds of NDCG are high (5 and 10). The gaps between the proposed algorithm and others are relatively huge on ImageNet dataset on all metrics.

In order to give an approximate comparison of the operating cost of each model, Table 4 and 5 present the model sizes (in megabytes) and the time taken to evaluate of a query pair (in miliseconds) on MSLR-30K and ImageNet dataset respectively. It is obvious that Random Forest (RF) models require a significant amount of storage, which is on the order of hundreds of megabytes. The second heaviest model is S-ELM, which consumes dozens of megabytes. The proposed HeMLGOP models are very compact, accounting for few hundreds of kilobytes, ranking second in terms of compactness.

Random Forest and S-ELM are not only heavy-weight but also slow compared to other algorithms during inference. While the proposed HeMLGOP models are not the fastest, they are relatively fast compared to other neural network based algorithms and much faster than the ensemble of trees. While the amount of storage can give a reasonable estimate on the compactness, the actual time taken to evaluate a query pair only illustrates a rough estimate on how fast each algorithm is during inference. The total number of computation is not necessarily proportional to the actual time taken, which is dependent on the implementation details such as caching or concurrency support.

5. CONCLUSION

In this work, we proposed an efficient algorithm to learn compact heterogeneous architectures of GOP-based networks. We employed these data-driven networks for solving ranking and content-based retrieval problems. The empirical results show that the proposed algorithm performs similarly or better than the related ensemble models, while requiring much smaller storage space and being faster for the ranking and retrieval task.

6. REFERENCES

[1] Y. Zhu, G. Wang, J. Yang, D. Wang, J. Yan, J. Hu, and Z. Chen, "Optimizing search engine revenue in sponsored search," in *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pp. 588–595, ACM, 2009.

[2] A. Karatzoglou, L. Baltrunas, and Y. Shi, "Learning to rank for recommender systems," in *Proceedings of the 7th ACM conference on Recommender systems*, pp. 493–494, ACM, 2013.

[3] M. Amini, N. Usunier, and C. Goutte, "Learning from multiple partially observed views-an application to multilingual text categorization," in *Advances in neural information processing systems*, pp. 28–36, 2009.

[4] J. Yu, D. Tao, M. Wang, and Y. Rui, "Learning to rank using user clicks and visual features for image retrieval," *IEEE transactions on cybernetics*, vol. 45, no. 4, pp. 767–779, 2015.

[5] C. J. Burges, "From ranknet to lambdarank to lambdamart: An overview," *Learning*, vol. 11, no. 23-581, p. 81, 2010.

[6] C. C. de Sá, M. A. Gonçalves, D. X. Sousa, and T. Salles, "Generalized broof-l2r: a general framework for learning to rank based on boosting and random forests," in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 95–104, ACM, 2016.

[7] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini, "Quickscorer: A fast algorithm to rank documents with additive ensembles of regression trees," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 73–82, ACM, 2015.

[8] D. Cohen, J. Foley, H. Zamani, J. Allan, and W. B. Croft, "Universal approximation functions for fast learning to rank: Replacing expensive regression forests with simple feed-forward networks," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pp. 1017–1020, ACM, 2018.

[9] G. Cao, A. Iosifidis, K. Chen, and M. Gabbouj, "Generalized multi-view embedding for visual recognition and cross-modal retrieval," *IEEE transactions on cybernetics*, vol. 48, no. 9, pp. 2542–2555, 2018.

[10] G. Cao, A. Iosifidis, M. Gabbouj, V. Raghavan, and R. Gottumukkala, "Deep multi-view learning to rank," *arXiv preprint arXiv:1801.10402*, 2018.

[11] S. Kiranyaz, T. Ince, A. Iosifidis, and M. Gabbouj, "Progressive operational perceptrons," *Neurocomputing*, vol. 224, pp. 142–154, 2017.

[12] R. H. Masland, "Neuronal diversity in the retina," *Current opinion in neurobiology*, vol. 11, no. 4, pp. 431–436, 2001.

[13] T. Qin and T. Liu, "Introducing LETOR 4.0 datasets," *CoRR*, vol. abs/1306.2597, 2013.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255, Ieee, 2009.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[16] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *Journal of machine learning research*, vol. 9, no. Aug, pp. 1871–1874, 2008.

[17] A. Liaw, M. Wiener, *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[18] C. P. Chen and Z. Liu, "Broad learning system: an effective and efficient incremental learning system without the need for deep architecture," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 1, pp. 10–24, 2018.

[19] S. Chatterjee, A. M. Javid, M. Sadeghi, P. P. Mitra, and M. Skoglund, "Progressive learning for systematic design of large neural networks," *arXiv preprint arXiv:1710.08177*, 2017.

[20] H. Zhou, G.-B. Huang, Z. Lin, H. Wang, and Y. C. Soh, "Stacked extreme learning machines," *IEEE transactions on cybernetics*, vol. 45, no. 9, pp. 2013–2025, 2015.