Universal Approximation Functions for Fast Learning to Rank

Replacing Expensive Regression Forests with Simple Feed-Forward Networks

Daniel Cohen*, John Foley*, Hamed Zamani, James Allan and W. Bruce Croft
Center for Intelligent Information Retrieval
University of Massachusetts Amherst
{dcohen,jfoley,zamani,allan,croft}@cs.umass.edu

ABSTRACT

Learning to rank is a key component of modern information retrieval systems. Recently, regression forest models (i.e., random forests, LambdaMART and gradient boosted regression trees) have come to dominate learning to rank systems in practice, as they provide the ability to learn from large scale data while generalizing well to additional test queries. As a result, efficient implementations of these models is a concern in production systems, as evidenced by past work.

We propose an alternate method for optimizing the execution of learned models: converting these expensive ensembles to a feed-forward neural network. This simple neural architecture is quite efficient to execute: we show that the resulting chain of matrix multiplies is quite efficient while maintaining the effectiveness of the original, more-expensive forest model. Our neural approach has the advantage of being easier to train than any direct neural models, since it can match the previously-learned regression rather than learn to generalize relevance judgments directly.

We observe CPU document scoring speed improvements of up to 400x over traditional algorithms and up to 10x over state-of-the-art algorithms with no measurable loss in mean average precision. With a GPU available, our algorithm is able to score every document in a batch in parallel for another 10-100x improvement. While we are not the first work to observe that neural networks are efficient as well as being effective, our application of this observation to learning to rank is novel and will have large real-world impact.

ACM Reference Format:
Daniel Cohen*, John Foley*, Hamed Zamani, James Allan and W. Bruce Croft. 2018. Universal Approximation Functions for Fast Learning to Rank. In *Proceedings of The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, Ann Arbor, MI, USA, July 8–12, 2018 (SIGIR '18),* 4 pages.
https://doi.org/10.1145/3209978.3210137

SIGIR '18, July 8–12, 2018, Ann Arbor, MI, USA

ACM ISBN 978-1-4503-5657-2/18/07…$15.00
https://doi.org/10.1145/3209978.3210137

1 INTRODUCTION

Forest-based regression models are the leading approach to train a learning to rank model [9, 19, 22], especially in industry [3, 5, 26]. These models have numerous benefits: they can be trained directly based on observed gradients of traditional IR metrics, unlike other learning to rank approaches [15], and learning is fairly efficient while achieving state-of-the-art results.

The drawback of forest models for production systems is that the cost of prediction is quite high: the naive algorithm for executing a tree is interpretation. This means that for a forest of size T and depth d, an interpreter will visit $O(Td)$ nodes for every point, and at each node it must compare a feature value and branch on the result. Given that pipelining is the ubiquitous strategy to making modern CPU architectures fast, these branch-heavy models are a worst-case scenario. At every decision point, pipeline is flushed, and actual instruction-level parallelism and therefore throughput will be quite low. This limits both query throughput and query latency of a learning to rank server, using a lot of machines and resources.

Efficiency of learning to rank approaches has drawn a greater amount of interest in recent years [19]. Before that work, Asadi and Lin argued that we should train these models to be more runtime-aware [1]. For many years now, researchers have been pursuing the question "How do we minimize the runtime cost of forest-based learning models?" [1, 4, 8, 13, 14, 19, 21, 25]. In this work, we propose answering this question by a key observation about the nature of these ranking ensembles.

Our key observation is that any ranking model will produce scores, given a set of document features as input. Once such a model is trained and validated, our goal for ranking is to output the predictions of that model as fast as possible. Therefore, if we had available to us a black box that could produce the same scores but faster, then we would be effectively executing our learned model. Hornik identifies feed-forward neural networks as valid *universal function approximators* [11]. In practice, this means that we can take these popular ranking ensembles and fully approximate them.

In this work, we first present analytic arguments for the effectiveness of feed-forward neural networks as full-approximators of regression forests (§3). Next, we provide an empirical demonstration of this technique: showing that we can learn approximations with no loss in mean average precision for LambdaMART ensembles trained on the MSN30k dataset, and for a Random Forest ranker trained on MQ2007 and trained on GOV2 (to demonstrate generalizability under more train/test skew). Simultaneously, we demonstrate the difficulties of directly training the same neural

* These authors contributed equally.

model on document judgments which reflects that using a generalized model (LambdaMART) leads to a generalized neural model. Finally, we present a brief sketch of our performance gains and observe a 2-10x improvement (Table 2) over previous published results.

In some sense, the core task we propose is *not novel* as it was possible and known since the introduction of the XOR-problem in 1969 alongside the perceptron [24], and confirmed for our specific functions later [11, 18]. However, recent works on faster algorithms for learning to rank ensembles (e.g., [19]) suggest that our revisiting of this theoretical work and empirical confirmation *is of significant research value* and will lead to important discussions in the learning to rank and information retrieval communities. We hope that industrial production systems employing such models can use our techniques to reduce their energy footprint while still satisfying users.

2 RELATED WORK

Our related work section is brief because we address the majority of related literature inline.

The Universal Approximation Theorem (UAT), proven by Hornik, shows that a single layer neural network under certain conditions is capable of fully approximating a continuous function [11]. While neural models did not gain traction in other fields for over a decade, this foundational contribution provides the base of current state-of-the-art neural architectures. Recently, Kraska et al. propose learning index structures to replace b-tree nodes for static on-disk indexing [17]. they make a case for approximating expensive tree-based functionality with cheaper, feed-forward neural networks, but they do not make the connection to learned trees.

In the realm of IR, Dehghani et al. show that approximating BM25 as a form of weak supervision is an effective way to train a feed-forward model for retrieval [10]. However, their work shows that while relevance is learned, the model only achieves parity with the signal function by including additional information not contained within the domain of BM25.

3 METHODOLOGY

The crux of this work comes from the UAT shown by Hornik [11], where the authors prove that a single layer neural network of sufficient width can approximate any continuous function within a set of conditions. Thus given N such that

$$N(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + b)$$

and assuming σ is a Lebesgue function, and $\int_a^b |\sigma(x)|^p dx < \infty$, for any $f \in L^p(K)$ $(p \geq 1)$ where K is a compact set in $\mathbb{R}^d$, then $\exists\ N$ for some $\epsilon > 0$ such that

$$||N - f||_{K,p} < \epsilon \tag{1}$$

This theorem has been applied in [10] where they copy the BM25 function, which satisfies the above conditions on finite collections. However, the same advantage that allows forest-based models to perform well for ranking, dividing the space into splits, also causes the relevance function, $s(\mathbf{x})$, to be piecewise continuous. This violates the UAT assumption, and presents the condition where f fails to converge pointwise at c. Thus

$$||N(c^-) - s(c^+)|| > ||s(c^-) - s(c^+)|| > \epsilon$$

We leverage work by Llanas et al. [18] that shows a piecewise continuous f can be approximated via a series of single layer neural networks within some ϵ. Thus, only each segment must satisfy the constraints proposed in Equation (1), which all current forest-based methods fulfill. Our empirical results demonstrate that ϵ is acceptable in practice.

This leads to N approximating the approximate function of a random forest within some error bound ϵ via a multilayer feed-forward neural network. As we can treat the domain of s as finite on a set of finite documents, this results in a compact subset of $\mathbb{R}$.

3.1 Neural Model Architectures

We use a four layer neural network with hidden dimensions [2000, 500, 500, 100] and a two-layer network with hidden dimensions [500,100]. We use ReLU6 as the non-linear activation. While the universal approximation theorem was shown with the sigmoid function, ReLU6 is continuous and bounded, which satisfies the assumptions set forth in [11]. In addition, ReLU6 has been shown to be robust to low precision operations, allowing for greater speedup in production if needed [12].

We use mean squared error as the loss function, with a batch size of 5000. Adam [16] is used to optimize the parameters with a learning rate of 0.001. While Llanas et al. [18] empirically show that a neural model is able to fully approximate s, they do so over the entire domain. We attempt to broaden our learned model from the training data by synthetically generating samples for which to train N. For each batch size of 5000, 2500 samples are created by randomly sampling around the discontinuous points of s in order to fully approximate the L2R function s.

Those familiar with neural networks used for ranking will note that we are using a "pointwise" learning and ranking style, which is often considered to be weaker than pairwise or listwise learning, but since we are learning from a pre-existing function (our existing forest model) rather than relevance, this pointwise model is more than sufficient.

3.2 Generating Random Training Data

In addition to leveraging our learned forest predictions on our training data, we also generate points from the learned model in order to guarantee that our system is matching the shape learned by the forest.

First, we create an empty list of points for each of the D features in our training set. We initialize these lists with the upper and lower bounds of each feature based on the training data. Then, we walk over every branch in our trained trees, and add each of these split points into our list of points for the appropriate features. At this point, we have identified where all of the discontinuities occur in the feature spaces of our piecewise model. We sort these discontinuity points, and replace them with ordered midpoints.

Now, generating a training point x is a simple as selecting random midpoints of interest for each feature, and evaluating the result using the original ensemble.

4 EXPERIMENTAL SETUP

We trained our LambdaMART and Random Forest models using the implementation available in RankLib [7]. This toolkit was preferred to others (e.g., XGBoost) because it accepted TREC-formatted relevance judgments and data without modification.

The MSN30k learning to rank dataset[1] is a commonly-used benchmark for the efficiency of ranking ensembles. We use this data for ease of comparison to reported numbers in past [19] and future work. However, a few observations we made about this dataset led us to believe that we should create our own secondary dataset rather than use an additional industry-released set of features.

For our second dataset, we explore our own version of the LETOR[2] dataset that is built from GOV2 and MQ2007. We do this for a variety of reasons that end up making our evaluation more robust. We naturally provide the extracted features, trained models, and the code used to run our experiments for future work[3].

4.1 Our Extracted Features

Although a complete listing of our features and code for extraction is provided in our source release, we give an overview of the features used here. We used web-based quality features [2], and common retrieval models [23] across the title, body, and document fields available to us in the GOV2 collection as parsed by the JSoup Java library. These features are quite similar in spirit to the original features for the LETOR set and similar to the MSN30k features, however we use a wider variety of retrieval models that require more sophisticated combinations.

5 RESULTS

Table 1: Effectiveness of L2R function and N approximation. There is no significant difference in mean average precision scores with $p < 0.05$. While most works using such datasets focus on early metrics like NDCG@10, we use mean average precision because it is a deeper measure to better show the effectively lossless nature of our approach. Significant differences are marked with an asterisk.

Method	# Layers	MSN30k MAP	GOV2 MAP
Regression Forest	-	0.6004	0.2995
N_{approx}	4	0.5950	0.2995
N_{approx}	2	0.5955	0.3007
$N_{\text{relevance}}$	4	0.5639*	0.2531*

5.1 Retrieval Effectiveness

As seen in Table 1, the neural model is able to almost completely approximate LambdaMart on MSN, while achieving parity on the GOV2 evaluation. The small difference in performance can be attributed to the stochastic nature of training deep neural models, and past work has shown that neural models have numerous local optima that closely approximate the global optimum [6].

Additionally, directly training N on the true relevance labels, referenced as $N_{\text{relevance}}$, results in significantly worse effectiveness. This can be attributed to the interpretation that relevance is not a function; for the same query, an identical document may be relevant for one user while non relevant for another. This results in the neural model fitting a new function rather than approximating a true functions, and reflects the success found in [10].

Examining the performance during training, N_{approx} in fact generalizes better than the forest based model in 93% of epochs prior to convergence. Thus while some accuracy is sacrificed, this results in improved performance on held out samples prior to completely over fitting on the training data and can be viewed as an additional form of regularization on the forest based model if desired.

5.2 Efficiency

While we refrained from re-implementing the Quickscorer algorithm [19], since it is undergoing a patent process[4], we present a brief sketch here that demonstrates our claim that feed-forward neural networks are much more efficient than forest based models, even without moving to GPU computation.

We therefore constructed an artificially low-baseline for our approach: Python/Tensorflow-CPU implementation of the same network on laptop hardware. We run these baselines on a Lenovo T430 laptop, with an Intel i5-3230M CPU @ 2.60GHz, and 16GB of RAM. And we compared directly to publication numbers available in the Quickscorer paper [19], which was run on a machine with an i7-4770K clocked at 3.50Ghz, with 32GiB RAM. Although the python-numpy version of the baseline is especially competitive with QuickScorer for very large forests we are able to significantly improve speed in comparison to the heavily engineered QuickScorer for large forests. Our GPU-based implementation is run on via PyTorch on a single NVIDIA TITAN X (Maxwell) GPU.

Our CPU-based implementation achieves a speedup of between 2-10x on the larger regression forests studied. Larger batches are more efficient, and this is especially true of our GPU implementation, which manages to score documents in under a microsecond on average for either our small or large network case.

Our dramatic improvements have the potential to change the story of production re-ranking, which is likely to be dominated by the branching needed to execute forest ensembles. With GPU scoring, there is essentially no cost to executing expensive models. We note that our GPU timing numbers include the time needed to transfer feature vectors from main memory to GPU memory, which shows the advantage of such a simple model.

We acknowledge that there are newer extensions of QuickScorer that focus on limiting tree count [21], exploting SIMD instructions [20], and potentially other optimizations are always possible. However, these papers present at most 3-5x improvements over the QuickScorer algorithm while being significantly more complex than our approach which provides an order of magnitude improvement for larger ensembles over the blockwise-Quickscorer.

We are confident that our results will carry over to production systems in terms of both latency and throughput. Whether these

[1] https://www.microsoft.com/en-us/research/project/mslr/
[2] https://www.microsoft.com/en-us/research/project/letor-learning-rank-information-retrieval/
[3] https://github.com/jjfiv/ltr2net
[4] https://github.com/hpclab/quickscorer

Table 2: Efficiency Comparison. All numbers refer to time to score an individual document in μs. Bounds represent the set of previously reported means and 5th and 95th percentiles for our models. Our simple numpy python approaches become competitive with large-tree executions of the QuickScorer algorithm. Production ranking models are likely to gain greatly in efficiency from our approach.

		1000 Trees		20,000 Trees	
Impl	Source	8 Leaves	64 Leaves	8 Leaves	64 Leaves
Generated C++ for Forest	If-Then-Else	8.2-10.3	55.9-55.1	709.0-772.2	4462.0-4809.0
QuickScorer [19]	QS	2.2-4.3	9.5-15.1	40.5-41.8	343.7-425.1
Blockwise-Quickscorer	BWQS	Unreported		33.5-40.5	236.0-274.7
Documents to re-rank (batch size):		100	200	500	1000
Tensorflow-CPU 4-layer	N	64.4-66.5	56.9-60.2	52.3-54.0	51.1-53.3
Tensorflow-CPU 2-layer	N	14.1-16.7	9.29-11.25	6.49-7.98	5.83-7.04
PyTorch GPU 4-layer	N	0.528-0.786	0.530-0.546	0.620-0.662	0.976-1.01
PyTorch GPU 2-layer	N	0.305-0.324	0.308-0.321	0.323-0.325	0.323-0.335

scoring systems have GPUs available for high-throughput or lean on the SIMD instructions in modern CPUs, the elimination of branching from our approach is an improvement that will generalize to architectures of the future.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we show that a forest based learning to rank function can successfully be approximated by a series of matrix multiplies in the form of a small neural network. This is supported both by theoretical guarantees and by empirical examination. Furthermore, the approximation results in slightly improved generalization in some cases which suggests that we have proposed a relatively safe method to improve efficiency without negatively impacting performance.

ACKNOWLEDGEMENTS

This work was supported in part by the Center for Intelligent Information Retrieval, in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA) via AFRL contact #FA8650-17-C-9116 under subcontract #94671240 from the University of Southern California. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon, and in part by NSF grant #IIS-1617408. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

REFERENCES

[1] Nima Asadi and Jimmy Lin. 2013. Training efficient tree-based models for document ranking *(ECIR)*. 146–157.
[2] Michael Bendersky, W Bruce Croft, and Yanlei Diao. 2011. Quality-biased ranking of web documents *(WSDM)*. 95–104.
[3] Microsoft Research Blog. 2015. RankNet: A ranking retrospective. https://www.microsoft.com/en-us/research/blog/ranknet-a-ranking-retrospective/. (2015).
[4] Gabriele Capannini, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, and Nicola Tonellotto. 2016. Quality versus efficiency in document scoring with learning-to-rank models. In *Information Processing & Management*. Elsevier, 1161–1177.
[5] Olivier Chapelle and Yi Chang. 2011. Yahoo! learning to rank challenge overview. In *Proceedings of the Learning to Rank Challenge*. 1–24.
[6] Anna Choromanska, Mikael Henaff, Michaël Mathieu, Gérard Ben Arous, and Yann LeCun. 2015. The Loss Surfaces of Multilayer Networks. In *AISTATS*.
[7] V Dang. 2015. RankLib, v.2.5-SNAPSHOT. https://sourceforge.net/p/lemur/wiki/RankLib. (2015).
[8] Domenico Dato, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. 2016. Fast ranking with additive ensembles of oblivious and non-oblivious regression trees. In *ACM Transactions on Information Systems (TOIS)*.
[9] Clebson CA de Sá, Marcos A Gonçalves, Daniel X Sousa, and Thiago Salles. 2016. Generalized BROOF-L2R: A General Framework for Learning to Rank Based on Boosting and Random Forests. In *SIGIR*. 95–104.
[10] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft. 2017. Neural Ranking Models with Weak Supervision. In *SIGIR*. 65–74.
[11] Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. In *Neural Networks*. 251–257.
[12] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. In *CoRR*. http://arxiv.org/abs/1704.04861
[13] Muhammad Ibrahim. 2017. Scalability and Performance of Random Forest based Learning-to-Rank for Information Retrieval. In *ACM SIGIR Forum*, Vol. 51. ACM, 73–74.
[14] Xin Jin, Tao Yang, and Xun Tang. 2016. A Comparison of Cache Blocking Methods for Fast Execution of Ensemble-based Score Computation. In *SIGIR*. 629–638.
[15] Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *SIGKDD*. 133–142.
[16] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. In *CoRR*. http://arxiv.org/abs/1412.6980
[17] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2017. The Case for Learned Index Structures. *arXiv preprint arXiv:1712.01208* (2017).
[18] Bernardo Llanas, Sagrario Lantarón, and Francisco J. Sáinz. 2008. Constructive Approximation of Discontinuous Functions by Neural Networks. *Neural Processing Letters* (2008), 209–226.
[19] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. 2015. Quickscorer: A fast algorithm to rank documents with additive ensembles of regression trees. In *SIGIR*. 73–82.
[20] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. 2016. Exploiting CPU SIMD extensions to speed-up document scoring with tree ensembles. In *SIGIR*. 833–836.
[21] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, and Salvatore Trani. 2017. X-DART: Blending Dropout and Pruning for Efficient Learning to Rank. In *SIGIR*. 1077–1080.
[22] Joel Mackenzie, J Shane Culpepper, Roi Blanco, Matt Crane, Charles LA Clarke, and Jimmy Lin. 2018. Query Driven Algorithm Selection in Early Stage Retrieval. In *WSDM*.
[23] Donald Metzler and W Bruce Croft. 2005. A Markov random field model for term dependencies. In *SIGIR*. 472–479.
[24] Marvin Minsky and Seymour Papert. 1969. Perceptrons.. In *MIT Press*.
[25] Xun Tang, Xin Jin, and Tao Yang. 2014. Cache-conscious runtime optimization for ranking ensembles *(SIGIR)*. 1123–1126.
[26] Hamed Zamani, Michael Bendersky, Xuanhui Wang, and Mingyang Zhang. 2017. Situational Context for Ranking in Personal Search. In *WWW*. 1531–1540.