Revealing the Secrets of Transformer-Based Learning to Rank: Techniques, Model Mechanisms, and Interpretability

Kartikesh Mishra

***Abstract*—In this paper, I present a comprehensive exploration of transformer-based learning-to-rank techniques, providing a deep dive into their principles, methodologies, and internal working. I discuss the foundational concepts of transformers and their advancements, focusing on their adaptation for learning to rank tasks. I explore the specific techniques and enhancements that have been developed to improve ranking performance in the machine learning domain as an end-to-end differentiable system. Additionally, I investigate approaches for gaining insights into the inner workings of the transformer models, enabling us to understand and interpret the ranking decisions made by these models. I highlight the significance of interpretability in learning to rank and discuss methodologies for visualizing and analyzing the behavior of transformers. Furthermore, I proposed a novel method to extract the rankings from a ranking learned representation. This paper serves as my class final project and can be taken as a comprehensive resource, offering a detailed overview of learning to rank models especially one using the transformer, its techniques, interpretability aspects.**

***Index Terms*—Ranking, Machine Learning, Neural Net, Rank Optimization, Learning-to-rank, Interpretability, Transformer Architecture, Deep Dive, Gradient-based Ranking**

I. Introduction

Ranking algorithms have a long history predating the advent of machine learning. They are closely intertwined with sorting algorithms, as ranking involves determining the positions of entities in sorted order within the entity space. This relationship between sorting and ranking will be further explored in the Terminology section (Section III).

Throughout the years, a plethora of sorting algorithms have been developed, each offering its own advantages and exhibiting specific performance characteristics. For a comprehensive overview of sorting algorithms, one can refer to the Wikipedia article on sorting algorithms [1]. This article provides an in-depth exploration of various sorting techniques, including well-known approaches such as Bubble Sort, Insertion Sort, Quick Sort, and Merge Sort. These algorithms serve as fundamental building blocks for generating rankings or ordered lists in diverse applications.

However, it is important to note that while sorting algorithms excel at efficiently arranging elements in a specific order, they are not inherently designed to address the unique challenges posed by ranking tasks. Traditional sorting algorithms prioritize time complexity and are often non-differentiable, lacking the ability to provide gradient information for end-to-end optimization in differentiable models.

In recent years, with the rise of machine learning and the demand for end-to-end differentiable models, the need for differentiable ranking operators has become more apparent. Differentiable ranking operators enable seamless integration of ranking operations into differentiable models, opening up new possibilities for optimizing models that involve ranking tasks. These tasks can range from learning-to-rank scenarios to incorporating ranking-based metrics into loss functions.

In the field of learning to rank, various models and approaches have been proposed to tackle the challenges of ranking. Section 2 provides an overview of related work in this area, exploring the strengths and limitations of existing techniques. I delve into different models that have been developed, including pointwise, pairwise, and listwise approaches, and discuss their applications in different domains. By understanding the landscape of existing approaches, we can build upon their foundations and explore novel techniques for transformer-based learning to rank.

To facilitate understanding and consistency in our discussions, Section 3 serves as a terminology guide. I define and explain the basic non-standard symbols and functions used throughout the paper, ensuring clarity in the mathematical notations employed in subsequent sections.

Section 4 focuses on differentiable sorting and ranking, as proposed by Blondel et al. [2] I delve into the details of their approach, which leverages the concept of differentiable sorting networks to enable end-to-end optimization of ranking tasks. By exploring the differentiable properties of sorting and ranking, I lay the groundwork for our investigation into transformer-based ranking techniques.

In Section 5, I discuss the importance and rise of transformers and present the architecture of my one-layer transformer model, specifically tailored for ranking tasks. I detail the components of the architecture, including embeddings, positional encodings, self-attention mechanisms, and multi-layer perceptrons. The training process, evaluation metrics, and results obtained from training our one-layer transformer model for pair-wise ranking are also presented.

Building upon the results obtained, the subsequent section focuses on the mechanistic interpretability analysis of the model's internal workings. I propose a novel method for extracting rankings from learned representations, enabling us to gain deeper insights into the decision-making process of the transformer-based ranking model. By visualizing and analyzing the learned representations, i aim to unravel the factors and patterns influencing the ranking decisions made by the model.

Finally, I discuss the future of this project, highlighting

potential areas for further research and development. I consider the challenges and opportunities in applying transformer-based learning-to-rank techniques to real-world scenarios, addressing issues such as scalability, interpretability, and adaptability to different domains. By offering a comprehensive exploration of transformer-based learning-to-rank techniques, including interpretability aspects and practical applications, this paper aims to provide researchers and practitioners in the field with valuable insights and guidance for effectively utilizing transformer models in the context of learning to rank.

II. Related Works

In the realm of machine learning (ML), researchers have explored different techniques and architectures to implement ranking systems. The goal is to develop models that can effectively order items or instances based on their relevance or importance. One of the early works in this domain is the OPRF (Ordinary Polynomial Regression for Pattern Recognition) algorithm introduced by Author et al. [3]. While this work primarily focuses on pattern recognition, it lays the foundation for using polynomial regression for ranking tasks, which aligns with the core principles of machine learning.

Over the years, several ML-based ranking algorithms have been proposed to address the challenges of learning-to-rank tasks. These algorithms aim to learn a ranking function that maps input data to an ordered output. Some notable approaches include Staged Logistic Regression (SLR) [4], which employs a staged training process to optimize the logistic regression model for ranking, and Non-Metric Optimization (NMOpt) [5], which explores non-metric optimization techniques for learning-to-rank problems.

As the field progressed, researchers introduced advanced ranking techniques that incorporated neural networks and optimization strategies. RankNet [6], for instance, employs a neural network architecture to learn pairwise preferences and uses a pairwise cross-entropy loss function for optimization. RankBoost [7], on the other hand, focuses on boosting-based ranking methods, where weak rankers are iteratively combined to create a strong ranking model.

LambdaRank [6] further extends the pairwise approach by incorporating information retrieval (IR) metrics. It uses the change in IR metric caused by swapping two items to modify the pairwise loss function in RankNet [6]. ListNet [8], a listwise ranking algorithm, treats the ranking problem as a permutation-based classification task and employs a softmax-based loss function to optimize the model.

In recent years, the focus has shifted towards differentiable ranking functions that enable end-to-end differentiable systems in machine learning architectures. Fast differentiable surrogates for ranking proposed by Author et al. [2] provide a notable advancement in this direction. These surrogates accurately capture the desired ranking metrics and scale effectively for large lists, enabling seamless integration into larger ML architectures. The end-to-end differentiable systems allow for joint optimization of ranking objectives and other task-specific objectives, leading to enhanced performance across various applications such as information retrieval, recommendation systems, and natural language processing.

To this date, there have been numerous ML architectures that have demonstrated success in ranking tasks. These include FaceNet [9], XGBoost [10], SAS-Rank [11], PRM [12], GVN-Rank [13], PiRank [14], Setrank [15] These examples highlight the diverse range of ML techniques and their effectiveness in addressing ranking challenges.

FaceNet [9] takes a different approach to ranking by focusing on face recognition. It utilizes deep convolutional networks to learn face embeddings and then employs the triplet loss function to train the model for ranking face images based on their similarity. FaceNet has demonstrated exceptional performance in face verification and identification tasks, showcasing the effectiveness of deep learning for ranking-related problems.

XGBoost [10] is a powerful gradient boosting framework that supports various ranking objectives and evaluation metrics. It leverages decision trees as weak learners and employs an optimized gradient boosting algorithm to construct an ensemble model. XGBoost has gained popularity due to its scalability, efficiency, and ability to handle large-scale ranking problems.

SAS-Rank [11] represents a recent advancement in learning to rank using a combination of Simulated Annealing and Evolutionary Strategy techniques. It explores both implicit and explicit learning approaches and incorporates relevance labels for improved ranking performance. SAS-Rank demonstrates the potential of leveraging metaheuristic algorithms for learning to rank.

SetRank [15] introduces a set-based ranking approach that considers the relationships and interactions among items in a set. It utilizes graph convolutional networks to capture the dependencies among items and learn their relative importance within the set. SetRank has shown promising results in scenarios where the ranking of a set of items is more relevant than individual item rankings.

These examples represent the continuous efforts to explore and innovate in the field of learning to rank. As the demand for personalized and context-aware ranking systems grows, researchers continue to develop new models and algorithms that cater to specific domains and user requirements. These advancements in learning to rank techniques pave the way for more effective and tailored ranking solutions in various applications.

While the field of learning to rank continues to advance, a significant challenge lies in understanding the internal workings of these models and their interpretation of the ranking function. Recent efforts in interpretability tasks like looking inside the circuits of neural networks [16], particularly in small deep learning models, have shed light on the understanding of model internals. Nanda et al. [17] train a small transformer model for modular addition and reverse engineer the model to uncover the actual algorithmic implementation. In this paper, a key goal is to lay the foundation for a similar interpretability framework for learning to rank models.

By exploring the techniques and internal workings of transformer-based learning to rank models, this research contributes to the understanding and interpretability of these models. The continuous development of personalized and context-

aware ranking systems relies on advancements in learning to rank techniques. This research sets the stage for more effective and tailored ranking solutions in various applications.

III. Basic Terminology

In this section, I define all non-standard basic symbols and functions I will be using throughout the paper. The term entity in this paper refers to any mathematical object. and the whole entity space is denoted by $\mathbb{E}$.

Definition 3.1 (Order Transformer Function): For any finite subset E = $\{e_1, e_2, ..., e_n\} \subset \mathbb{E}$ and a permutation of $[n]$, $\sigma = (\sigma_1, \sigma_2, ..., \sigma_n)$, the order transformation function $T_n : \mathbb{E}^n \times [n]! \to \mathbb{E}^n$ is defined by $T_n(E, \sigma) = \{e_{\sigma_1}, e_{\sigma_2}, ..., e_{\sigma_n}\}$ Here, $[n]!$ denotes the set of all possible permutations of $[n]$.

By employing the order transformation function, we can rearrange the elements in the subset E according to the given permutation σ. This allows us to generate different orderings of the elements within E and explore their implications in ranking and sorting operations. Moreover, the order transformation function $T_n(E, \sigma)$ is a linear operation. Clearly, $\forall i \in [n], e_{\sigma_i} = E.\alpha_i$ where α_i is a vector s.t. only σ_i element is 1 and rest all 0. We can construct a $n \times n$ permutation matrix M_σ with α_i as its ith row i.e. $M_\sigma[i, \sigma_i] = 1 \forall i \in [n]$ and all other element in M_σ is 0. From the construction, $T_n(E, \sigma) = M_\sigma E$

Definition 3.2 (Ordered Entity Set): An entity set E = $\{e_1, e_2, ..., e_n\}$ is called an ordered entity set if $e_1 \geq e_2 \geq ... \geq e_n$ and totally ordered entity set if $e_1 > e_2 > ... > e_n$

Definition 3.3: A permutation σ is called sorter of E, represented by σ_E if $T_n(E, \sigma)$ is an ordered entity set.

Lemma 3.1: For any $E \subset \mathbb{E}, \sigma_E$ exists and is unique if $T_n(E, \sigma_E)$ is a totally ordered entity set.

Since E is a finite set, by well ordering principle, its bounded and has atleast one maxima e_{m_1} for some $m_1 \in [n]$, next we do the same on the set E - $\{e_{m_1}\}$ to find e_{m_2} and so on. By this construction, σ_E exists and $\sigma_E = \{m_1, m_2, ..., m_n\}$. Furthermore, if $T_n(E, \sigma_E)$ is totally ordered entity set then we do not have ties of maxima at all and at every stage, the maxima is unique and thus only one possibility of σ_E

Definition 3.4: The descending sorting operation can be defined by

$$\text{sort}(E) = T_n(E, \sigma_E)$$

Definition 3.5: Similarly, the descending ranking π_E can be defined as the inverse of sorter permutation σ_E in the permutation space.

$$T_n(\sigma_E, \pi_E) = \{n, n-1, ..., 1\}$$

For the ascending order, we can pass -E.

IV. Differentiable Ranking

Designing differentiable ranking operators presents challenges due to the inherent non-differentiability of the ranking function. The ranking operator behaves as a piecewise constant function, with derivatives being null or undefined. This non-differentiability complicates the integration of ranking operations into differentiable models that heavily rely on gradient-based optimization methods like backpropagation.

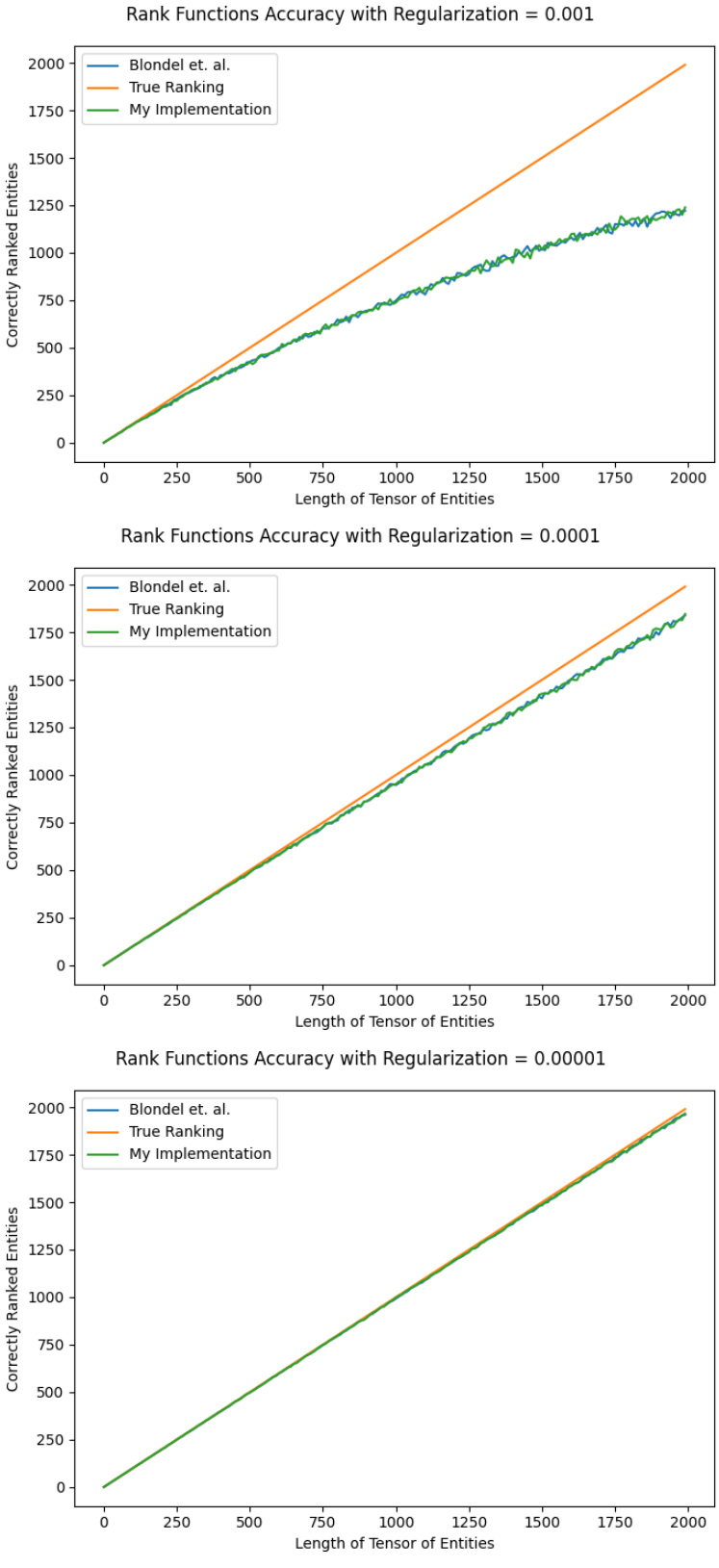

Fig. 1. Rank Functions comparison

Most of the existing approaches to differentiable ranking often rely on approximations or proxies, which can sacrifice computational efficiency compared to traditional ranking algorithms. Although these approximations provide differentiable counterparts to ranking, they often suffer from time complexity limitations, making them less practically useful.

In their work, Blondel et al. [2] propose a novel approach to differentiable ranking that overcomes the limitations of existing approximations. Their approach tackles both time complexity and computational efficiency concerns by constructing differentiable ranking operators using projections onto the permutahedron, which represents the convex hull of permutations. Additionally, they leverage a reduction to isotonic optimization to ensure exact computation and differentiation.

By developing differentiable ranking operators with improved time and space complexity, we can enable the seamless integration of ranking operations into differentiable models. This advancement not only improves optimization efficiency and accuracy but also enhances interpretability by providing deeper insights into the ranking process within the model. I have re-implemented the approach by Blondel et al. [2], specifically optimizing it for PyTorch tensors to leverage GPU capabilities. In Figure 1, I provide a brief comparison of my implementation with their implementation and true rankings using L2 regularization.

V. Ranking using Transformer

The Transformer architecture [18] has emerged as a groundbreaking model in natural language processing or in general any sequence based learning tasks , showcasing its capabilities in tasks such as machine translation, language generation, and text classification. With recent advancements in models like ChatGPT and GPT-4, Transformers have gained significant attention and proving the capability of the models to an extent it can be fooled as a human.

Transformers revolutionize traditional sequence processing by introducing self-attention mechanisms that capture global dependencies between input tokens, enabling better modeling of contextual relationships. This self-attention mechanism allows the model to attend to different parts of the input sequence adaptively, providing a more comprehensive understanding of the context.

The effectiveness of Transformers in capturing complex dependencies and producing high-quality representations has motivated researchers to explore their potential in ranking tasks. One notable attempt in this direction is the Transformer-based Personalized Re-ranking Model (PRM) [12], which leverages the power of Transformers to learn ranking functions.PRM extends the Transformer architecture to accommodate ranking tasks by incorporating ranking-specific components and training strategies.

A. Model Architecture

The architecture, as illustrated in Figure 2, is a single layer transformer and the model I am using to learn ranking pairwise consisting of several key components:

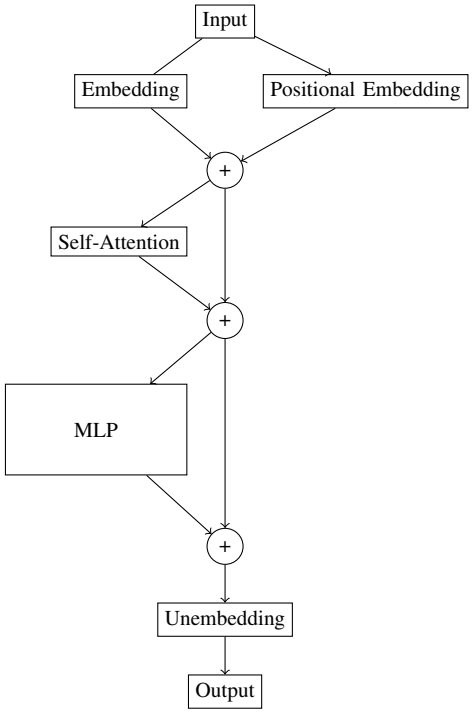

Fig. 2. One-layer Transformer model

*1) **Embedding**:* The ranking input consists of a set of entities, each represented by an embedding. These embeddings capture the semantic meaning of the entities and allow the model to learn their relationships. To account for the order of entities, a positional embedding is added to the input embeddings. This positional information enables the model to understand the sequential relationships between entities.

From here onwards, I will refer Embeddings as sum of Embedding and Positional Embedding. This is implemented by a matrix W_e of size num_entities x embedding_dim. For an input sequence of the entities X, we get their embedding representation as $X_e = W_e[X]$.

2) Attention: The attention layer plays a crucial role in capturing the interdependencies between entities in the input sequence. It computes attention scores that measure the relevance or importance of each entity with respect to others.

Query, Key, and Value Representations: The attention mechanism operates on three different representations: query, key, and value. These representations are derived from the input embeddings and are obtained by multiplying the input embeddings with corresponding weight matrices.

Query: $Q = X_e \cdot W_Q$, where Q is the query matrix, and W_Q is the query weight matrix of size embedding_dim x query_dim.

Key: $K = X_e \cdot W_K$, where K is the key matrix, and W_K is the key weight matrix of size embedding_dim x key_dim.

Value: $V = X_e \cdot W_V$, where V is the value matrix, and W_V is the value weight matrix of size embedding_dim x value_dim.

Attention Scores: The attention scores quantify the relevance or similarity between the query and key representations. This is achieved by computing the dot product between the

query and key matrices, followed by an appropriate scaling factor to control the magnitude of the attention scores:

$$S = \frac{Q \cdot K^T}{\sqrt{h}}$$

Here, S represents the attention score matrix, and h = key_dim is the dimension of the key representation. In fact, h = query_dim = key_dim = value_dim.

Attention Weights: The attention scores are transformed into attention weights by applying a softmax function across the rows (or columns) of the score matrix to ensure that the weights sum up to 1 for each query entity:

$$A_W = \text{softmax}(S)$$

The matrix W represents the attention weight matrix, where each row corresponds to a query entity, and the values within each row denote the attention weights assigned to the key entities.

Weighted Value Representations: The attention weights are used to weight the value representations, allowing the model to focus on relevant information during the aggregation step. This is accomplished by multiplying the attention weights with the value matrix:

$$C = A_W V$$

Here, C represents the context matrix, which captures the aggregated information based on the attention mechanism.

The output of the attention layer is the context matrix C, which contains the aggregated information from the input sequence, considering the interdependencies between entities.

*3) **MLP (Multi-Layer Perceptron)**:* After the self-attention layer, a multi-layer perceptron (MLP) is applied to further transform the representations. The MLP introduces non-linear transformations to enhance the model's capacity to capture complex patterns and relationships in the ranking data. It consists of a fully connected neural network, which comprises multiple layers of interconnected units. These units can learn complex mappings and capture higher-level features from the input representations..

The transformation applied by the MLP can be mathematically represented as follows:

$$X_M = ReLU(W_M(X_e + C))$$

In this equation, W_M represents the weight matrix of the fully connected neural network in the MLP layer. The matrix multiplication $(W_M \cdot (X_e + C))$ applies the learned weights to the concatenated input, resulting in a transformed representation. The ReLU activation function ReLU$(\cdot)$ introduces non-linearity by applying element-wise rectified linear activation to the transformed values. This non-linear activation helps the model capture complex patterns and relationships in the ranking data.

By incorporating the MLP layer after the self-attention layer, the model can learn more expressive representations and better capture the intricate dependencies and patterns in the ranking data, ultimately improving its ranking performance.

*4) **Unembedding**:* At the final stage of the model, the transformed representations are passed through an unembedding layer. This layer maps the transformed representations back to the entity space, allowing the model to produce ranking scores which can be converted into probabilities using softmax. The unembedding layer acts as a decoder, mapping the learned representations back to the original entity space and returns $U_e(X_M + X_e)$ where U_e is the unembedding matrix.

B. Dataset Preparation

To prepare the dataset, I started by creating a large pool of truth values represented by a 500 x 64 random matrix D_T. This matrix captures the underlying truth values of 500 entities across 64 attributes and remains constant throughout the experiment.

Next, for a given num_city = n and num_attributes = m, I constructed a dataset $\mathbf{D}$ of size $n^2m \times 3$, where n represents the number of cities and m represents the number of attributes. The dataset $\mathbf{D}$ includes all permutations of two entities across an attribute. In other words, each entry $[A, B, C] \in \mathbf{D}$ corresponds to the question "Is entity A bigger than entity B across attribute C?"

The labels of the dataset are derived from the truth matrix D_T. For each entry $[A, B, C] \in \mathbf{D}$, the label Y is determined by comparing the corresponding values in D_T. Specifically, the label Y is defined as $Y = \{D_T[C, A] > D_T[C, B] : [A, B, C] \in \mathbf{D}\}$, representing whether entity A is larger than entity B across attribute C.

C. Experimental Setup

To assess the performance of the proposed one-layer Transformer model for ranking tasks, a series of experiments were conducted. A total of 20 different model and dataset pairs were created, varying the values of num_entities, num_attributes, and embedding_dim.

For each model, the training process involved utilizing binary cross entropy as the loss function. The training dataset consisted of 70% of the available data, while the remaining 30% was used for validation purposes and the batch size was set to 512. The AdamW [19]optimizer was employed, with the following hyperparameter settings: learning rate = 1e-3, weight decay = 0.2 (tuned for optimal results), and betas = (0.9, 0.98). Additionally, a scheduler was utilized to control the learning rate during training.

$$\text{BCE}(Y, \hat{Y}) = -\frac{1}{N}\sum_{i=1}^{N}[Y_i \log(\hat{Y}_i) + (1 - Y_i)\log(1 - \hat{Y}_i)]$$

Here, Y represents the true labels, $\hat{Y}$ represents the predicted labels, N denotes the total number of samples in the batch i.e. 512. The loss function measures the dissimilarity between the predicted values and the true labels, penalizing larger discrepancies and thus fits my model objective.

The one-layer Transformer models were generated using the transformer_lens library [20], which provided a convenient framework for implementing and training the models.

Figure 3 illustrates the progression of loss, accuracy, and consistency metrics during the training phase. The models were trained for a total of 200 epochs, allowing for a comprehensive evaluation of their performance and convergence behavior.

D. Results

The training progress was monitored using various metrics, including loss, accuracy, and consistency. The consistency metric measured the model's ability of model to learn for any $[A, B, C] \in \mathbf{D}\}$, $(D_T[C, A] > D_T[C, B]) + (D_T[C, B] > D_T[C, A]) = 1$ Remarkably, the one-layer Transformer models demonstrated implicit learning of this property, as depicted in the consistency graph.

The loss graph and accuracy graph depicted the model's training progress over epochs, showcasing how it gradually improved its ranking performance and achieved high accuracy on the test datasets. These results demonstrate the potential of Transformer-based models for ranking tasks and highlight the effectiveness of the proposed architecture. The model with highest overall accuracy measured as frac_train*(train_accuracy) + (1- frac_train) * (validation_accuracy), referred as best model onwards, achieved the 99.71% accuracy. An attempt was made to recover the ground truth rankings and compare it with actual rankings. Since the model is a pairwise ranking model, I used merge sort [1] algorithm to convert this pair-wise algorithm to rank the whole entities. the best model were able to match 76.41% of the exact ranks averaged on 5 attributes and the average spearmann correlation coeffient was found 0.7864.

VI. Analysis of the Model Internals

In this section, I delved into the models internals to understand how ordering is handled by transformers. In doing so, I propose a novel method to extract rankings from a learned representation. Suppose an entity subset $E = \{e_1, e_2, ..., e_n\} \subset \mathbf{E}$ is represented by nxd matrix M i.e. each entity $\in E$ is a d-size vector and M is the learned representation. I call this method, rank probe and define it as ranking of weighted projection of entities representation M on a given attribute representation A (which is also vector of size d) i.e. for any subset matrix $\hat{M} \subset M$

$$R(\hat{M}, W) = rank(\hat{M}@(W. * A))$$

where ,

$$W = argmin_{w \in \mathbb{R}^d, \hat{M} \subset M} L(R(\hat{M}, w), R_A(\hat{M}))$$

L here is a loss function can be derived from either Spearman's rank correlation coefficient or Kendall Tau and $R_A(M)$ is the ground truth ranking of M with respect to attribute A. The rank function [2] used here is a differentiable function and thus, we can train a neural network to find optimal W. For this paper, I ran an experiment to trained this probing using 1 - Spearman's rank correlation coefficient as the objective function i.e.

$$L(R_1, R_2) = \frac{6 \sum (R_1 - R_2)^2}{n(n^2 - 1)}$$

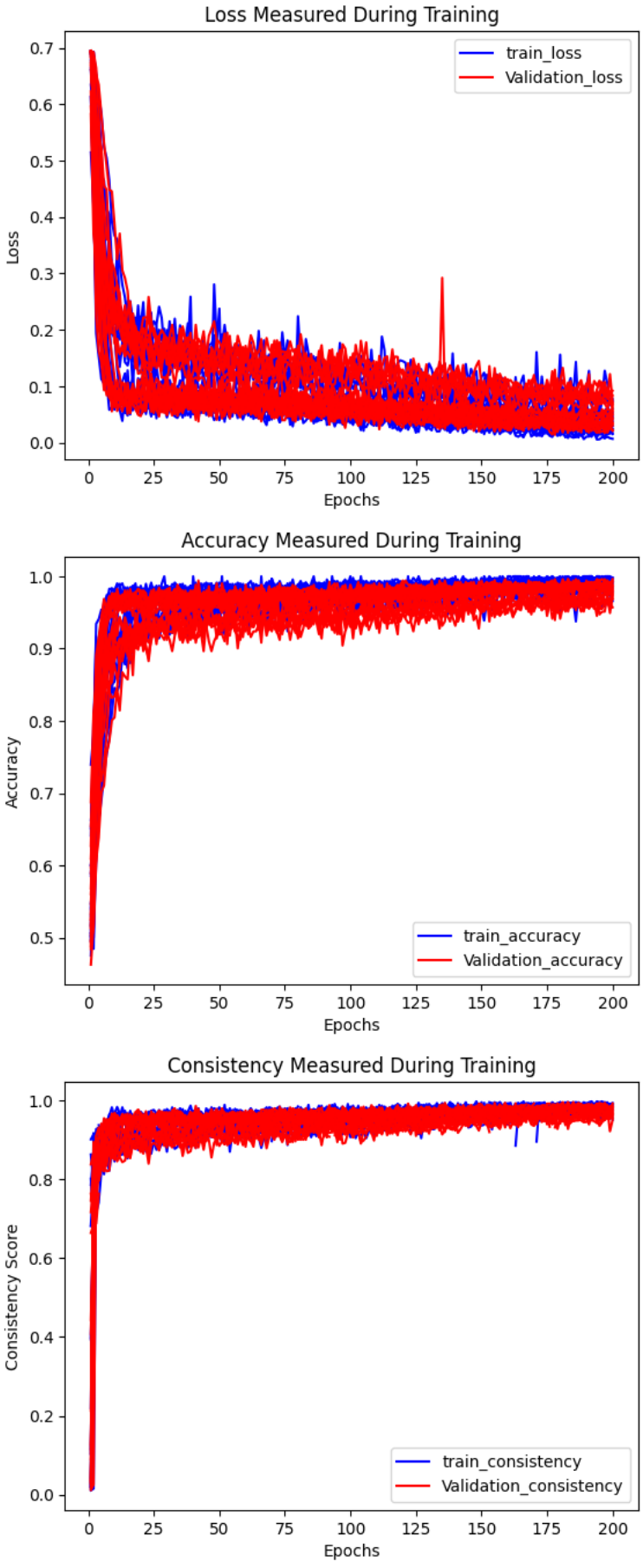

Fig. 3. Loss, Accuracy and Consistency during Training

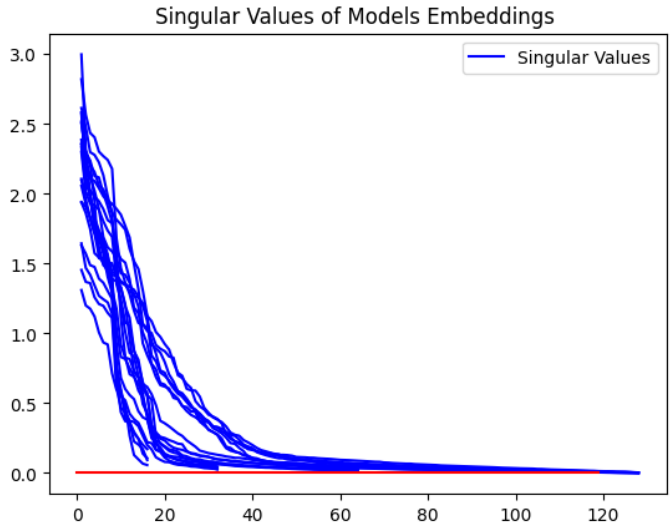

Fig. 4. Singular Values of Learned Model Embedding

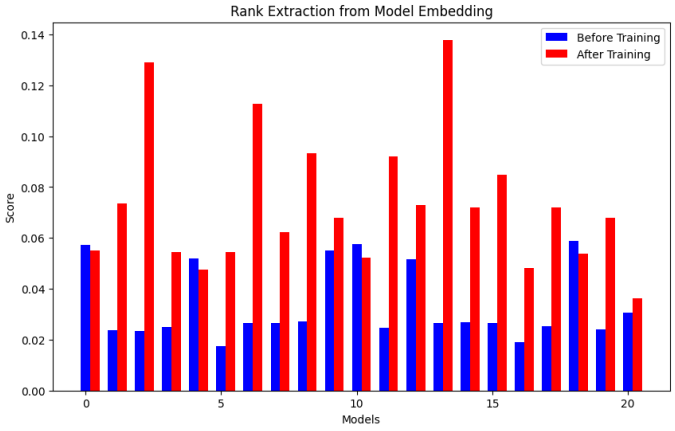

Fig. 5. Rank Extraction from Models Embeddings

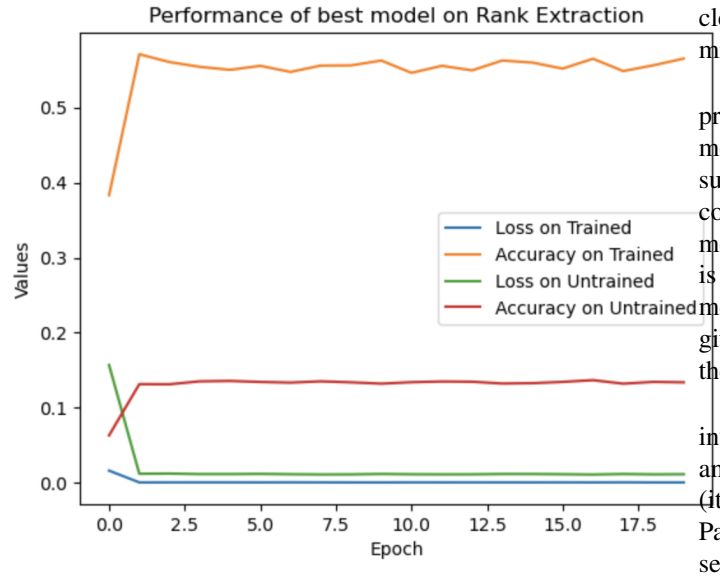

Fig. 6. Rank Extraction on the best Model

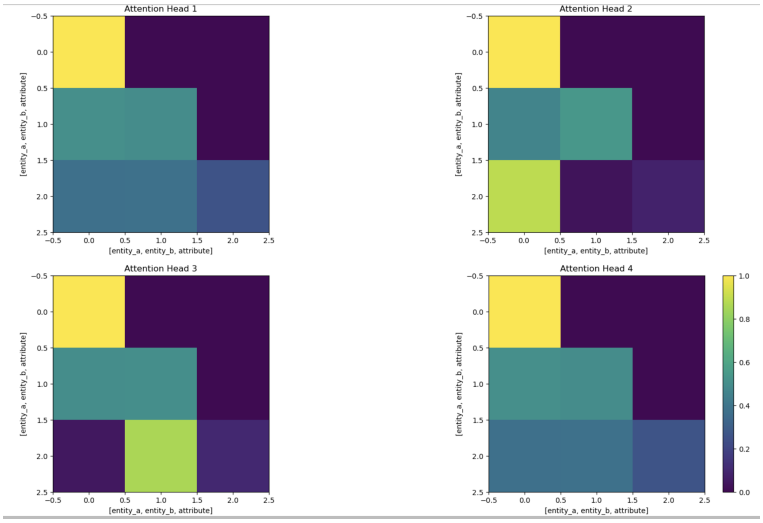

Fig. 7. Attention Patterns in the model Internals

where n = number of entities.

The motivation behind keeping a weight comes from the singular value decompositions of the embedding matrix of models. As depicted in figure 4, shows not all embedding dimensions gives useful insight.

As can be seen from the expression, the parameter space for W is quite large and thus can be very extensive task computationally requiring huge computational resources. I randomly sampled 100 subsets of M and trained the separate rank probes for 10 epochs on the embedding matrix of models both before and after the training of the model. The result is shown in figure 5 which though shows significant improvement in case of most of the models while for few, the after training score was slightly off than the before training ones. This could be due to some factors like the embedding matrix for those models got initialised really well as can be seen they have higher pre-training scores than other models. Another major factor can be just that the 100 samples are too low to get the clear picture as total possible subsets are $2^{\text{num\_entities}}$ and the minimum num_entities used was 130.

To further verify my proposed method, I trained two rank probe before and after the training on the best performing model for 20 epochs using num_entities^2 randomly sampled subset. As can be seen i the figure 6, the loss values shows conversion towards 0 for both pre-trained and post-trained model embeddings but, the gap between performance in them is quite significant and the rank probe trained on post-trained most embeddings was able to match 57% of exact rank and given that the model was able to do so for 76.41%, this justifies the working of the rank probing.

One more experiment I ran to gain insights of the models internals is to look for attention patterns passing all data points and averaging them. I used 4 attention heads in my models (it was kept constant throughout the experiment). Attention Patterns (a distribution of probability distribution over input sequence) can reveal where in the input the models looks into give the resulting output. The attention patterns shown in figure 7 highlights that the first and last attention heads give

equal attention to entity_a, entity_b and the attribute while the second pays almost all attention to the entity_a and the third pays almost all attention to the entity_b. This further verifies my experiment to run the rank probing on the model embeddings as it seems that all the representations related task is handled by the embeddings and the attention is used for model to allow selecting the desired values and compare throughout the rest of the model.

VII. Conclusion

In this paper, I have presented a comprehensive exploration of transformer-based learning-to-rank techniques. I have discussed the foundational concepts of transformers and their adaptation for ranking tasks. Through an in-depth analysis of related works, I have learned various models and techniques employed in the field of learning to rank. Additionally, I have focused on the importance of interpretability in learning to rank and proposed a novel method for extracting rankings from learned representations.

My experiments using the one-layer Transformer model have demonstrated its effectiveness in ranking tasks. The model, trained using binary cross entropy loss, has shown promising results across different datasets with varying dimensions. The use of the AdamW optimizer and a scheduler for controlling the learning rate has further enhanced the model's performance.

The results of my experiments highlight the potential of transformer-based models for ranking tasks in machine learning. The ability to capture complex patterns and relationships, coupled with end-to-end differentiability, provides a powerful framework for addressing ranking challenges. Furthermore, my proposed method for interpretability analysis adds another dimension to understanding and interpreting the decisions made by the model. While I showed some evidence that the transformers internals works by handling ranking aware representations in the embedding space and extracting it using attention and comparing them to give the prediction, it remains open to test the hypothesis on the large language models at the scale of ChatGPT and GPT4.

Overall, this paper contributes to the growing body of knowledge in transformer-based learning-to-rank techniques. By providing a comprehensive overview of the principles, methodologies, and internal workings of these models, as well as exploring their interpretability aspects, I aim to empower researchers and practitioners in the field to leverage the potential of transformers for ranking tasks. I believe that this work opens up new avenues for improving ranking performance and encourages further exploration in this exciting area of research.

References

[1] Wikipedia, "Sorting algorithm," Wikipedia: The Free Encyclopedia, 2023, accessed: 15 May 2023. [Online]. Available: https://en.wikipedia.org/wiki/Sorting_algorithm

[2] M. Blondel, O. Teboul, Q. Berthet, and J. Djolonga, "Fast differentiable sorting and ranking," in *International Conference on Machine Learning*. PMLR, 2020, pp. 950–959.

[3] N. Fuhr, "Optimum polynomial retrieval functions based on the probability ranking principle," *ACM Transactions on Information Systems (TOIS)*, vol. 7, no. 3, pp. 183–204, 1989.

[4] W. S. Cooper, F. C. Gey, and D. P. Dabney, "Probabilistic Retrieval Based on Staged Logistic Regression," in *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1992.

[5] B. T. Bartell, G. W. Cottrell, and R. K. Belew, "Automatic Combination of Multiple Ranked Retrieval Systems," in *SIGIR'94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, organised by Dublin City University*. Springer London, 1994.

[6] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to Rank using Gradient Descent," in *Proceedings of the 22nd International Conference on Machine Learning*, Microsoft Research, One Microsoft Way, Redmond, WA 98052-6399, 2005.

[7] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *Journal of machine learning research*, vol. 4, no. Nov, pp. 933–969, 2003.

[8] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 129–136.

[9] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[10] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[11] O. A. S. Ibrahim and E. M. Younis, "Hybrid online–offline learning to rank using simulated annealing strategy based on dependent click model," *Knowledge and Information Systems*, vol. 64, no. 10, pp. 2833–2847, 2022.

[12] C. Pei, Y. Zhang, Y. Zhang, F. Sun, X. Lin, H. Sun, J. Wu, P. Jiang, J. Ge, W. Ou *et al.*, "Personalized re-ranking for recommendation," in *Proceedings of the 13th ACM conference on recommender systems*, 2019, pp. 3–11.

[13] O. A. S. Ibrahim and E. M. Younis, "Combining variable neighborhood with gradient ascent for learning to rank problem," *Neural Computing and Applications*, pp. 1–12, 2023.

[14] R. Swezey, A. Grover, B. Charron, and S. Ermon, "Pirank: Scalable learning to rank via differentiable sorting," *Advances in Neural Information Processing Systems*, vol. 34, pp. 21 644–21 654, 2021.

[15] L. Pang and et al., "SetRank: Learning a Permutation-Invariant Ranking Model for Information Retrieval," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020.

[16] C. Olah, N. Cammarata, L. Schubert, G. Goh, M. Petrov, and S. Carter, "Zoom in: An introduction to circuits," *Distill*, vol. 5, no. 3, pp. e00 024–001, 2020.

[17] N. Nanda, L. Chan, T. Liberum, J. Smith, and J. Steinhardt, "Progress measures for grokking via mechanistic interpretability," *arXiv preprint arXiv:2301.05217*, 2023.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[19] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[20] N. Nanda, "Transformerlens," 2022. [Online]. Available: https://github.com/neelnanda-io/TransformerLens