Stabilizing Neural Search Ranking Models

Ruilin Li[1,2], Zhen Qin[2], Xuanhui Wang[2], Suming J. Chen[2], Donald Metzler[2]
[1]Georgia Institute of Technology, Atlanta, GA, USA
[2]Google LLC, Mountain View, CA, USA
ruilin.li@gatech.edu,{zhenqin,xuanhui,suming,metzler}@google.com

ABSTRACT

Neural search ranking models, which have been actively studied in the information retrieval community, have also been widely adopted in real-world industrial applications. However, due to the high non-convexity and stochastic nature of neural model formulations, the obtained models are *unstable* in the sense that model predictions can significantly vary across two models trained with the same configuration. In practice, new features are continuously introduced and new model architectures are explored to improve model effectiveness. In these cases, the instability of neural models leads to unnecessary document ranking changes for a large fraction of queries. Such changes lead to an inconsistent user experience and also adds noise to online experiment results, thus slowing down the model development life-cycle. How to stabilize neural search ranking models during model update is an important but largely unexplored problem. Motivated by trigger analysis, we suggest balancing the trade-off between performance improvements and the number of affected queries. We formulate this as an optimization problem where the objective is to maximize the average effect over the affected queries. We propose two heuristics and one theory-guided method to solve the optimization problem. Our proposed methods are evaluated on two of the world's largest personal search services: Gmail search and Google Drive search. Empirical results show that our proposed methods are highly effective in optimizing the proposed objective and are applicable to different model update scenarios.

KEYWORDS

learning to rank, neural network, trigger analysis

ACM Reference Format:
Ruilin Li[1,2], Zhen Qin[2], Xuanhui Wang[2], Suming J. Chen[2], Donald Metzler[2]. 2020. Stabilizing Neural Search Ranking Models. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3366423.3380030

1 INTRODUCTION

Ranking is an important problem in many real-world applications such as web search [36], recommender systems [10], and personal search [8]. Due to the recent success of deep learning, neural search ranking models are not only being extensively studied in academic research, but also deployed in many large-scale industrial applications [3, 23–27, 37].

To improve user engagement and achieve business goals like revenue growth, it is common for practitioners to continuously iterate on the quality of search ranking models. A typical workflow for incremental model development is to first propose a quality improvement idea, such as adding new informative features or changing the model architecture, and then to train an updated model. After validating the quality of the model offline, one runs controlled online experiments (e.g. A/B testing) on real traffic and conducts statistical analysis to evaluate the performance of the updated model. Traditional controlled experiments can be noisy due to various factors (e.g. systematic variance in user behavior) and typically require weeks or longer to converge with statistically significant results. Hence, trigger analysis [13, 14] has been proposed and widely used in practice [46] to facilitate more efficient online experimentation. In particular, trigger analysis only computes metrics for queries that trigger the treatment (i.e., experimental) model. In the context of ranking, the treatment is triggered if the updated model generates a different ranked list of results than the base model. Trigger analysis allows performance changes to be isolated to the set of affected queries, which results in a better signal-to-noise ratio and makes it easier to achieve statistically significant differences between the treatment and control. This helps accelerate the development cycle and allows more accurate measurement of the performance impact over affected queries.

Through repeated experimentation, we noticed that trigger analysis is not as useful for neural search ranking models as such models tend to introduce a significant amount of result set churn even if trained over the same training data. This is because neural networks are known to be highly non-convex and hence admit a large number of local minima. Moreover, the training of a neural network involves randomness, such as random weights initialization and stochastic gradients in training. These factors make it difficult to effectively apply trigger analysis to neural ranking models.

To demonstrate this effect, we train ten identical neural ranking models (i.e., using the same hyper-parameters and data set) on Gmail search data. They perform comparably in terms of standard ranking metrics such as mean reciprocal rank (MRR). However, Figure 1 shows that the top-6 results returned for a vast majority of queries differ across pairs of these models. We refer to this as the **instability** issue in training neural search ranking models.

Given this observation, it should not be surprising that the same phenomenon happens when updating neural ranking models. Queries that are affected (more formally defined in subsection 3.2) due to instability are undesirable since they significantly weaken the power of trigger analysis. Also, significant churn in the result set can lead to an inconsistent user experience [41]. For example, a user may be accustomed to seeing one set of results for a given query, but that order may change as a result of model instability

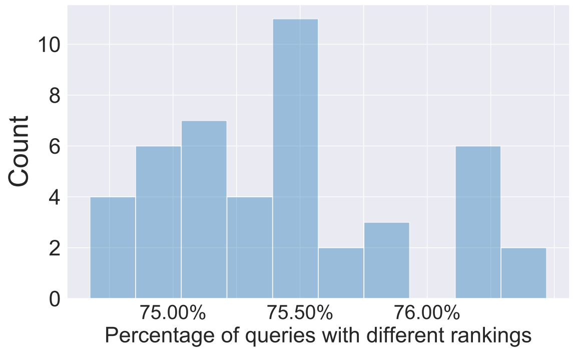
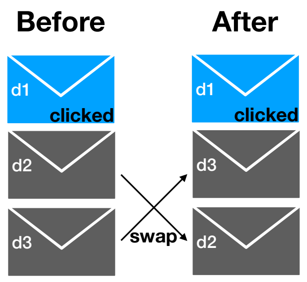
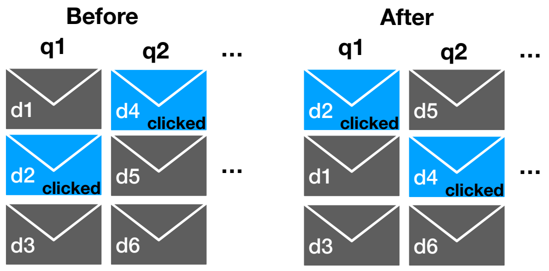


Figure 1: Histogram of percentage of affected queries with different rankings. For the 10 independent runs, we compare each pair of them, generating $\binom{10}{2} = 45$ data points. Each query is associated with up to 6 documents. The sample standard deviation of MRR is minimal (0.001).



(a) Individual level



(b) Population Level

Figure 2: Two typical types of unnecessary ranking changes. Figure 2a is at individual level and Figure 2b is at population level. Both types of changes do not improve the *overall* ranking metric (e.g., MRR). Blue represents clicked documents.

after the model is updated. As such, quality-neutral changes are often undesirable.

A natural question to ask is: when a new ranking model that improves ranking performance is trained and compared against a baseline model, how many of the changed rankings actually contribute to ranking metric improvements and how many are simply due to the instability of the neural ranking model training? It is desirable to keep only useful changes while suppressing unnecessary changes. However, it is not straightforward to detect unnecessary changes. As shown in Figure 2, while it is possible to detect and filter individual-level instability (e.g., using counterfactual logging), it is more challenging to do so for population-level instability.

In this paper, we propose three novel approaches to stabilize neural ranking models. The first two, additive boosting and latent cross, center around the idea of boosting in machine learning [17, 18]. These two methods are mainly targeted for incremental feature additions, which is very common in practice. The motivation for these two approaches is to boost the base model with the newly added features while avoiding training a new model from scratch. These two methods differ in the extent to which they allow interactions between new and existing features. The third approach, named score regularization, trains a new neural model but uses scores of the base model to regularize the loss function. Hence, it takes into account the changes relative to the base model. We will show that score regularization directly optimizes an approximation of the objective function we care about in trigger analysis. Moreover, score regularization is a unified approach that is suitable for both feature additions and model architecture changes.

To summarize, our main contributions are:

- We identify the instability problem in neural search ranking which has important real-world implications for more precise trigger analysis and maintaining a consistent user experience.
- We propose a single objective that is closely related to trigger analysis, balancing performance gains and the percentage of affected queries.
- We present three novel approaches to stabilize neural ranking model training. In particular, we show score regularization is a theoretically-grounded approach to optimize the objective we care about.
- We conduct extensive experiments using two of the world's largest personal search data sets, Gmail search and Google Drive search, to show the effectiveness of the proposed methods and to provide guidelines for practitioners.

2 RELATED WORK

Our work is rooted in neural search ranking models and online experimentation. There is an abundance of prior work in these areas and we review them in this section.

2.1 Neural Search Ranking Models

Learning-to-rank (LTR) [30] is the most widely-used and successful frameworks for developing ranking functions today. The framework, which encompasses so-called pointwise, pairwise, and listwise approaches, typically uses supervised machine learning techniques to learn parameterized ranking functions.

In some of the earliest work in the area, Burges proposed RankNet [5] to leverage the power of neural networks to improve search ranking models. RankNet is a pairwise model that was later extended to a listwise approach named ListNet [7]. The theoretical properties of the listwise approach were studied in [49].

Deep Structured Semantic Model (DSSM) [20] combines deep learning with latent semantic models. Convolutional Deep Structured Semantic Models (C-DSSM) [40] extends DSSM by introducing a CNN with max-pooling. The architecture of C-DSSM can extract the most salient local features from queries and documents to enhance overall semantic representation effectiveness. Convolutional Latent Semantic Model (CLSM) is built upon DSSM and further captures contextual information via a series of projections from one layer to another in a CNN architecture [39]. The distributed representation learned by DSSM and its variants can also be used to re-rank documents based on in-session contextual information [29]. Motivated by the dramatic progress in academia, there is also a substantial increase in deploying neural search ranking models in industry, e.g., LinkedIn[37], Microsoft[30], Walmart[3], and Google[44].

For a more detailed review of neural search ranking models, we refer readers to [19, 33, 35]. To the best of our knowledge, no existing work has studied the problem of stabilizing neural search ranking models.

2.2 Large Scale Online Experimentation

Consumer-facing companies, such as Amazon [26, 27], Microsoft [24, 25] and Google [46], use controlled experiments to evaluate changes affecting user experience and make decisions accordingly.

Conventional online controlled experiments such as A/B testing diverts incoming requests into control and treatment groups, evaluates the performance of the base and experiment models separately and then compares the two. This approach usually requires a large volume of traffic to obtain significant results hence can slow down the model development cycle. In practice, many online services have many features that only affect a small proportion of incoming traffic. In such cases, including requests that do not trigger the treatment dilutes the true effect. It is instead more efficient to analyze only requests that can trigger the treatment. This type of trigger analysis is efficiently implemented by counterfactual logging in practice [13, 14, 46]. By restricting comparison to queries for which base and experiment models produce different rankings, trigger analysis can considerably improve the sensitivity of online experiments.

Another data-efficient method is interleaved evaluation [9, 21, 22]. Interleaved evaluation merges two rankings into one interleaved ranking and reports win, lose, or tie, and hence provides a direct comparison between two rankers. Interleaved evaluation reduces individual-level noise (Figure 2a), but can not handle population-level noise (Figure 2b).

Although the design of our objective function is motivated by trigger analysis, our proposed methods can directly improve the stability of neural search ranking models and can be evaluated with any online experimentation method.

3 BACKGROUND

We provide some background on learning to rank and trigger analysis in this section. Key concepts are also defined here.

3.1 Modeling Preliminary

Suppose we have a set of N queries $Q = \{q_i\}_{i=1}^N$, each query q is associated with a group of D documents $\mathcal{D}(q) = \{d_j^q\}_{j=1}^D$ and each document $d \in \mathcal{D}(q)$ is given a relevance label $r(q, d)$ (e.g., a binary or graded label). For notational convenience, we collect relevance labels of documents associated with the same query q as a $D-$dimensional vector $\boldsymbol{r}(q)$. The goal of ranking models is to generate a ranking $\boldsymbol{\pi}(q) = (\pi_1, \pi_2, \cdots, \pi_D)$ (π_j is the rank of document d_j) over $\mathcal{D}(q)$ for each q to optimize an overall ranking metric $\mathbb{E}_{q\sim u(Q)}$ [M(q)], where $u(Q)$ is uniform distribution over Q and popular choices of ranking metrics M include reciprocal rank (RR) and normalized discounted cumulative gain (NDCG) [11].

Learning-to-rank (LTR) algorithms learn a parameterized function $f = f_{\boldsymbol{\theta}}(\cdot)$ which takes features of a query-document pair (q, d) as input, ranks documents associated with q according to the value of f, and outputs a ranking $\boldsymbol{\pi}(q) = \boldsymbol{\pi}(q, f)$. Due to the non-smoothness of commonly used ranking metrics, learning $\boldsymbol{\theta}$ by directly optimizing this objective can be difficult and approximation is often needed [4, 6, 7]. We adopt this approach and discuss in detail the ranking metric we use and its approximation in subsection 6.1.

3.2 Definition of Key Concepts

This subsection provides definitions for three key concepts that are frequently referred to in this paper.

- **Base model**: The neural ranking model before applying any update is denoted by f^{base}. This is typically the deployed model in a production system.
- **Benchmark model**: The default approach to applying model updates without considering stabilization, denoted by $f^{\text{benchmark}}$. Concretely, if the model update adds features, the default approach can be to simply concatenate existing features $\boldsymbol{x}_{\text{exist}}$ with new features $\boldsymbol{x}_{\text{new}}$, $\boldsymbol{x} = (\boldsymbol{x}_{\text{exist}}, \boldsymbol{x}_{\text{new}})$ and train $f^{\text{benchmark}}$ with the same neural net architecture as that of f^{base}. When the model update is an architecture change, one uses existing features and trains $f^{\text{benchmark}}$ with the new architecture. $f^{\text{benchmark}}$ will serve as a baseline for comparison purposes later.
- **Affected query**: A query for which the base model f^{base} and an experimental model f produce different rankings. All affected queries are denoted by $Q_T = \{q \in Q | \boldsymbol{\pi}(q, f) \neq \boldsymbol{\pi}(q, f^{\text{base}}\} \subseteq Q$.

3.3 Trigger Analysis

Trigger analysis refers to the study of the treatment's effect on affected queries Q_T. Trigger analysis is important in evaluating performance changes of an experimental model, since including queries not in Q_T not only dilutes the measured impact of the experiment, but also introduces noise caused by user variance into metric computation and hence lowers the power of statistical tests.

In statistics, the coefficient of variation is defined as CV $= \frac{\sigma}{\mu}$, where $\mu > 0$ is the mean and σ is the standard deviation of some statistic. It is commonly used to measure the dispersion of a probability distribution and is closely related to hypothesis testing. The smaller CV is, the easier it is to observe significant results in statistical tests. [28]. We now show, via the following theorem, that trigger analysis indeed lowers the coefficient of variation compared with controlled experiments.

Theorem 1. *The coefficient of variation of improved ranking metric ΔM in controlled experiments $CV_{control}$ is larger than that in trigger analysis $CV_{trigger}$.*

Proof. Note that when $q \in {Q_T}^c$, $\Delta M(q) = 0$. Combining this key fact and conditional expectation, we have

$$\begin{aligned}\mathbb{E}[\Delta M] &= \mathbb{P}(Q_T)\mathbb{E}[\Delta M|Q_T] + \mathbb{P}({Q_T}^c)\mathbb{E}[\Delta M|{Q_T}^c] \\ &= \mathbb{P}(Q_T)\mathbb{E}[\Delta M|Q_T]\end{aligned}$$

$$\begin{aligned}\mathrm{VAR}[\Delta M] &= \mathbb{E}[\Delta M^2] - \mathbb{E}[\Delta M]^2 \\ &= \mathbb{P}(Q_T)\mathbb{E}[\Delta M^2|Q_T] - \mathbb{P}(Q_T)^2\mathbb{E}[\Delta M|Q_T]^2 \\ &= \mathbb{P}(Q_T)^2\mathrm{VAR}[\Delta M|Q_T] + \mathbb{P}(Q_T)\mathbb{P}({Q_T}^c)\mathbb{E}[\Delta M^2|Q_T]\end{aligned}$$

Therefore,

$$\begin{aligned}\frac{\mathrm{CV}_{\mathrm{control}}}{\mathrm{CV}_{\mathrm{trigger}}} &= \frac{(\mathrm{VAR}[\Delta M])^{\frac{1}{2}}/\mathbb{E}[\Delta M]}{(\mathrm{VAR}[\Delta M|Q_T])^{\frac{1}{2}}/\mathbb{E}[\Delta M|Q_T]} \\ &= \sqrt{1 + \frac{1-\mathbb{P}(Q_T)}{\mathbb{P}(Q_T)}\frac{\mathbb{E}[\Delta M^2|Q_T]}{\mathrm{VAR}[\Delta M|Q_T]}} \\ &> 1\end{aligned}$$

□

In particular, we see from the above proof when $\mathbb{P}(Q_T)$ is small, trigger analysis can greatly improve the significance of the test.

The most straightforward way to implement trigger analysis is to divert traffic to either the control group or the experiment group conditioned on triggering. Since determining which query request triggers the treatment may require additional processing at runtime and increase latency, triggering cannot easily be implemented online. Instead, we run the experiment model on the logs of the control group and base model on the logs of the experiment group to identify the affected queries (i.e., Q_T). Due to its **counterfactual** nature, this approach is known as counterfactual logging [13, 46].

4 PROBLEM DESCRIPTION

Deep neural networks are wellsuited for large scale search ranking problems for several reasons. First, understanding the semantics of queries and documents is critical to the success of search engines and deep models are known for their ability to learn meaningful semantic embeddings [31, 32]. Second, ranking is typically a large-scale problem, while neural network models are highly scalable because training can be distributed and paralleled using specialized hardware [1, 12].

However, neural networks are highly non-convex and admit a large number of local minima. Recent research has shown that almost all local minima are also global minima [34, 42, 43]. Furthermore, multiple sources of randomness are involved in training deep models, including but not limited to random initialization, minibatch training, (asynchronous) distributed training, randomness of optimizers (in addition to stochastic gradient, e.g. Adagrad [16]). Therefore, even with exactly the same setup (data, model hyper-parameters, etc.), independent training runs of the same neural net result in different models. We refer to this phenomenon as **instability**.

Instability is also common when updating the neural ranking models for better performance. Often, when a modeling improvement is proposed, we observe improved overall ranking performance but a large number of affected queries. One natural question is, how many of these ranking changes are meaningful when it comes to improving ranking quality and how many of them are simply due to instability? From a practical perspective, a large percentage of affected queries weakens the power of trigger analysis. Inconsistent ranking results over time can also hurt the user experience. Therefore, we would like to answer the following research questions: (1) how do we identify useful and non-useful ranking changes? (2) is it possible to suppress unnecessary changes and only keep the useful ones, resulting in more stable neural ranking models?

In order to leverage the information of new features or the capacity of more powerful neural architectures, it is reasonable to expect some changes in ranking results that contribute to improved ranking quality. In contrast, unnecessarily affected queries do not contribute to performance gains. Motivated by this key observation, we propose to maximize the following objective

$$\frac{\mathbb{E}[\Delta M]}{\mathbb{P}(Q_T)} \tag{1}$$

where the numerator $\mathbb{E}[\Delta M]$ is the overall improvement in ranking metric M and the denominator is the size of affected queries. This quantity measures the mean improved ranking metric per affected query and we denote it by $\Delta M/\mathrm{aq}$.

Intuitively, we expect significant improvements for each affected query and hence unnecessary changes are discouraged. If equation (1) is maximized by $f^\star$, the instability issue no longer exists as

$$f^\star = \arg\min_f\{\mathbb{P}(Q_T(f))|\mathbb{E}[\Delta M(f)] = \mathbb{E}[\Delta M(f^\star)]\}$$

Among all models which perform equally well as $f^\star$, $f^\star$ affects the fewest queries and hence there are no unnecessary changes. In addition, the objective in equation (1) is also closely related to trigger analysis. One can rewrite the objective as the performance gain of the ranking metric from trigger analysis

$$\begin{aligned}\Delta M/\mathrm{aq} &= \frac{\mathbb{E}[\Delta M(q)]}{\mathbb{P}(Q_T)} = \frac{1}{|Q_T|}\sum_{q\in Q}\Delta M(q) \\ &= \frac{1}{|Q_T|}\sum_{q\in Q_T}\Delta M(q) = \mathbb{E}[\Delta M(q)|Q_T]\end{aligned}$$

When $\Delta M/\mathrm{aq}$ is optimized, we observe a stronger treatment effect per affected query, which potentially leads to more statistically significant results.

One concern is that $\Delta M/\mathrm{aq}$ is relative and as a result can be misleading. For example, if a new model affects only one query of the base model – the worst performing one, by lifting the clicked document to the top – the new model would show a very large improvement when utilizing trigger analysis, even though the new model is almost identical to the base one. Although being a sound and reasonable theoretical concern, very small model changes can affect millions of queries and users in practical systems. In our experiments, even the slightest model change gives rise to at least 10% affected queries. We therefore argue that the proposed objective is still meaningful to optimize in practice.

If optimization problem (1) can be solved successfully by $f^{\star}$ then we will be able to answer the aforementioned questions. First, essential changes are identified by $Q_T(f^{\star})$ and unnecessary changes are characterized by $Q_T(f^{\star})^{c}$. Second, unnecessary changes can be suppressed by maximizing trigger metric change.

One potential approach to optimizing $\Delta M/\text{aq}$ is to develop a mechanism (e.g., based on offline analysis) that determines whether the base model or the experimental model performs better for incoming queries. This approach can help ensure the ranking performance is improved for every affected query. However, both the base model and the updated model would need to be served simultaneously which may require non-trivial engineering effort. The additional logic required to implement this would also require additional processing time and hence has the potential to increases latency.

The next three subsections are dedicated to solving optimization problem (1).

5 METHODS

In this section, we introduce three methods to stabilize neural search ranking models. The first two methods are heuristics and mainly target cases where new features are added to a base model. The last method, score regularization, (approximately) solves the optimization problem $\max_f \Delta M/\text{aq}$ directly and can be applied to various model development scenarios.

5.1 Additive Boosting

As much of the instability is caused by re-training a deep neural model from scratch, the most intuitive approach to mitigate this issue is to avoid complete re-training and exploit the base model as much as possible.

Additive boosting is a classic family of boosting methods [17, 18] and is widely used in modern data science. We propose using additive boosting

$$f^{\text{boost}}((\boldsymbol{x}_{\text{existing}}, x_{\text{new}})) = f^{\text{base}}(\boldsymbol{x}_{\text{existing}}) + b_{\phi}(\boldsymbol{x}_{\text{new}})$$

In this approach, during model training, the base model is fixed. b_ϕ is a learnable booster that fits the residual between the prediction of the base model $f_{\text{base}}(\boldsymbol{x}_{\text{existing}})$ and the ground truth. b_ϕ can take different forms. For example, b_ϕ can be a linear model or even a neural network itself when a feature with more than one dimension (e.g., embeddings for a single feature) or a group of new features are added.

Additive boosting is particularly interesting for low coverage features, which is not uncommon in practice. Low coverage features (or feature groups) are those for which only a small fraction (e.g. less than 20%) of queries have documents with non-default feature values. When a low coverage feature is added, it is sensible to expect there to only be a few affected queries between the new model and the base model because the ranking should not change for queries whose associated documents have the same default value for the new feature. Additive boosting can be quite effective in such cases.

Futhermore, scores of documents that correspond to queries that only have default feature values get changed by the same constant. This type of score translation does not affect ranking. Therefore, $\mathbb{P}(Q_T)$ is naturally upper-bounded by the coverage of the (non-default-valued) new feature.

5.2 Latent Cross

One major drawback of additive boosting is that it does not allow higher order feature interactions between existing and new features, hence meaningful interactions between them can not be captured. We therefore propose using latent cross [2], a boosting-inspired idea that is able to learn second-order interactions between features.

Figure 3 provides a schematic illustration of boosting with latent cross. All variables in the base model f^{base} are frozen, similar to that in additive boosting. After being processed by a trainable dense layer for dimension matching, new features are then element-wise multiplied with the output of the second-to be-last layer in f^{base}. Boosting with a latent cross architecture can learn meaningful feature interactions between existing and new features, potentially providing better overall ranking performance than additive boosting. Intuitively, since the base model is fixed, the number of affected queries should be lower than when training from scratch, though we no longer ensure a tight upper bound for low coverage features as with additive boosting.

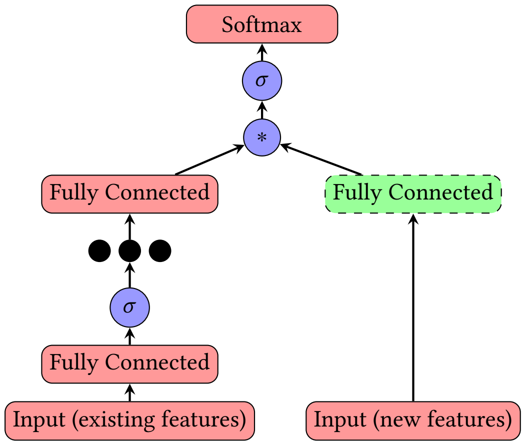


Figure 3: Illustration of latent-cross architecture. Dashed green and solid red boxes represent trainable and untrainable layers respectively. $*$ is element-wise multiplication and σ is an activation function (e.g., sigmoid, ReLU). Note the left part (base model) does not have to be a stack of fully connected layers.

Figure 4 shows the relative positions of additive boosting, latent cross, and the benchmark model on the spectrum of affected queries and feature interactions. Additive boosting and latent cross provide two alternatives to balance the trade-off between new/existing feature interactions (usually correlated with overall ranking performance) and the number of affected queries. Intuitively, it is easier to improve performance for a small subset of queries than to achieve the same improvement over a large subset, hence additive boosting and latent cross can achieve better results than the benchmark

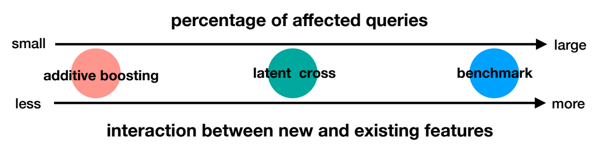

percentage of affected queries
small
large
additive boosting
latent cross
benchmark
less
more
interaction between new and existing features

Figure 4: An illustration of additive boosting, latent cross, and benchmark on the spectrum of affected queries and feature interaction.

model and can be regarded as heuristically optimizing the objective in equation (1).

Additive boosting and latent cross are limited to model updates that involve adding new features. They can not handle updates that involve architecture changes. This limitation motivates us to consider a more unified and systematic approach in the next section.

5.3 Score Regularization

Score regularization allows re-training a neural network model to optimize the objective function in (1) directly.

Typically, neural search ranking models are updated by adding more informative features or changing the model architecture to increase capacity. In both scenarios, the update results in a more powerful model and we may reasonably assume $\mathbb{E}[\Delta M(f)] > 0$. Commonly used ranking metrics (such as MRR, NDCG) are all bounded, therefore $\mathbb{E}[\Delta M(f)]$ is also upper bounded. Note that optimization problem (1) is a standard fractional programming and can be converted to an equivalent problem due to Dinkelbach [15],

$$\max_f \frac{\mathbb{E}[\Delta M(f)]}{\mathbb{P}(Q_T(f))} = \max_f \{\mathbb{E}[\Delta M(f)] - \lambda \mathbb{P}(Q_T(f))\}$$

for some $\lambda > 0$. The latter one can be written as

$$\begin{aligned} & \max_f \{\mathbb{E}[\Delta M(q,f)] - \lambda \mathbb{P}(Q_T(f)) \\ \iff & \max_f \{\mathbb{E}[M(f) - M(f^{\text{base}})] - \lambda \mathbb{P}(Q_T(f))\} \\ \iff & \max_f \{\mathbb{E}[M(f)] - \lambda \mathbb{P}(Q_T(f))\} \\ \iff & \min_f \underbrace{-\mathbb{E}[M(f)]}_{\text{first term}} + \lambda \underbrace{\mathbb{P}(Q_T(f))}_{\text{second term}} \end{aligned} \tag{2}$$

In practice, λ is unknown and treated as a tunable hyper-parameter. The first term is the original objective in LTR and is approximated the same way as in the benchmark model. The second term is non-smooth and also needs to be approximated. An intuitive method is to use the difference of scores as surrogate of $\mathbb{P}(Q_T(f))$. Compared with the benchmark model, the only difference is that the loss function has an additional score regularization term, hence we refer to this approach as score regularization. We denote the minimizer of equation (2) by f_λ^{sr}.

Compared with the benchmark method, score regularization differs only by the additional regularization term and can be easily implemented in practice.

The remainder of this section describes two possible approximation techniques and the connection to ensemble methods.

5.3.1 Pointwise Score Regularization. The first option is a direct comparison of document scores output by the base model and the new model:

$$\mathbb{P}(Q_T(f)) \approx \frac{1}{|Q|} \sum_{q \in Q} \sum_{d \in \mathcal{D}(q)} R(f(d), f^{\text{base}}(d))$$

where $R(\cdot,\cdot)$ is a bi-variate function. We refer to this choice as **pointwise** score regularization because the score regularization term drives the new model to reproduce the exact score of the base model for every document. Common choices of $R(\cdot,\cdot)$ include the L_2 norm (squared) and L_1 norm

$$L_2 : R(x,y) = |x-y|^2, \qquad L_1 : R(x,y) = |x-y|.$$

5.3.2 Listwise Score Regularization. Although pointwise score regularization is straightforward, it can be too restrictive in that any monotonic transformation (e.g. translation) can change scores but still preserve the ranking. Requiring new models to achieve a particular score configuration can be difficult.

As discussed in [7], the value of $f(q,d)$ is the logit of document d being clicked for query q. We therefore propose the following **listwise** score regularization

$$\mathbb{P}(Q_T(f)) \approx \frac{1}{|Q|} \sum_{q \in Q} R(\tilde{f}(q), \tilde{f}^{\text{base}}(q))$$

where $\tilde{f}(q) = \frac{1}{\sum_{d \in \mathcal{D}(q)} \exp\{f(q,d)\}} [\exp\{f(q,d_1)\}, \cdots, \exp\{f(q,d_D)\}]$ and $R : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$. With softmax normalization, we turn a logit vector into a probability vector and guarantee ranking invariance under a common type of monotonic transformation - translation. We consider four regularizers for listwise score regularization:

$$L_2 : R(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{D} (x_i - y_i)^2, \qquad L_1 : R(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{D} |x_i - y_i|,$$

$$\text{KL}: R(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{D} x_i \log \frac{x_i}{y_i}, \qquad \text{Hellinger}: R(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{D} (\sqrt{x_i} - \sqrt{y_i})^2.$$

5.3.3 Connection to Ensemble Methods. When $\lambda = 0$, the regularization term vanishes and the score regularization model f_0^{sr} degenerates to the benchmark model $f^{\text{benchmark}}$; when $\lambda = \infty$, the regularization term forces f_∞^{sr} to recover f^{base}. Therefore, when λ varies in $(0,\infty)$, f_λ^{sr} is a continuous (nonlinear) interpolation between f^{base} and $f^{\text{benchmark}}$ in model space. See Figure 5 for an illustration. In this sense, score regularization can be viewed as an ensemble of the base model and the benchmark model.

Surprisingly, score regularization is also closely related to boosting. In particular, when pointwise L_2 regularization is used, Shalev showed that under mild conditions, training a model with the loss in Equation 2 amounts to performing one *selfie-boost* step [38]. *Selfie-boost*, unlike conventional boosting algorithms such as gradient boosting trees [18], does not use a linear combination of generated models as its final model, but always replaces the old model with the new one and results in a single model in the end. Therefore, *selfie-boost* is well-suited for boosting neural nets. The *selfie-boost* point of view also helps us understand why score regularization should be

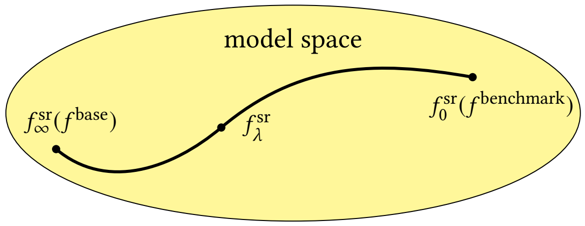

model space
$f^{\text{sr}}_\infty(f^{\text{base}})$
f^{sr}_λ
$f^{\text{sr}}_0(f^{\text{benchmark}})$

Figure 5: Illustration of f^{sr}_λ being a (nonlinear) interpolation between f^{base} and $f^{\text{benchmark}}$ in model space.

better than the benchmark in terms of ΔM/aq. Due to the regularization term, we have $\mathbb{P}(Q_T(f^{\text{sr}}_\lambda)) < \mathbb{P}(Q_T(f^{\text{benchmark}}))$ and thanks to the boosting effect, we have $\mathbb{E}[M(f^{\text{sr}}_\lambda)] > \mathbb{E}[M(f^{\text{benchmark}})]$, hence score regularization has better performance measured in ΔM/aq.

In summary, the score regularization approach is motivated to directly (but approximately) optimize ΔM/aq and works for a wide variety of different types of model changes. Hence it is a unified and systematic approach to stabilize neural search ranking models.

6 EVALUATION

In this section, we study how to stabilize neural ranking model updates in the context of personal search. We conduct a series of experiments over two commercial personal search engines, Gmail and Google Drive search, to evaluate the effectiveness of the methods proposed in section 5.

6.1 Experiment Setup

6.1.1 Data set and metric. Our experiments use click-through data sampled from Gmail and Google Drive search logs. We discard queries without clicks. In total, we have hundreds of millions of queries in each data set. Each query is associated with up to 6 documents, which is a constraint arising from the user interface. To protect user privacy, all data are anonymized using k-anonymity [45] and the only content feature we have access to are query and document n-grams that are frequent in the corpus. We use these content features, together with other categorical (e.g. `is_spam`, `is_social`, `is_promotion`, etc.) and dense numeric (e.g., `document_age`, `num_recipients`, etc.) features to build feature vectors for each query-document pair (q, d). Among collected queries, 80% are used for training, 15% are used for validation and hyper-parameter tuning, and the remaining 5% are held out for testing.

For personal search, click-through data serves as a binary indicator of relevance [21], For such binary relevance labels, we use mean reciprocal rank (MRR) as our primary metric of interest. If we denote the index of the clicked by $j^\star$ then

$$\text{MRR} = \mathbb{E}_{q\sim u(Q)}[\text{RR}(q)] = \frac{1}{|Q|}\sum_{q\in Q}\frac{1}{\pi_{j^\star}(q)}$$

We use MRR as the objective in learning-to-rank and approximate it via the softmax loss [7], which is a differentiable lower bound of MRR in log scale [4]. This approximation allows us to interpret the value of f as the logit of the document being clicked.

We primarily use weighted reciprocal rank (WRR) as the evaluation metric for our offline experiments and report mean improved weighted reciprocal rank per affected query (ΔWRR/aq).

$$M(q) = \text{WRR}(q) = \frac{w_{j^\star}}{\pi_{j^\star}(q)}$$

where $w_1, w_2, \cdots, w_D$ are constants that can be estimated using the *RandPair* method [48]. WRR accounts for position bias in clicks and bridges the gap between our offline experiments and actual online experiments [48]. We also report the relative improvement of overall WRR ($\Delta\text{WRR}\% = \frac{\mathbb{E}[\Delta\text{WRR}(f)]}{\mathbb{E}[\text{WRR}(f^{\text{base}})]}$) and the percentage of affected queries $\mathbb{P}(Q_T)$ for a deeper understanding of the model behavior.

6.1.2 Experiment Design. We use a multi-layer perceptron (MLP) neural net implemented in Tensorflow as our parameterized function f. Our ranking model is trained with the softmax cross entropy loss [4, 7]. We perform two major categories of experiments. The first category adds new features to a base model. We experiment with three groups of new features for both Gmail and Drive. The new features with their characteristics are summarized in Table 1. We cover typical practical feature addition scenarios, including adding a single feature (both low and high coverage) and groups of features. The second category changes the model architecture. Here, we experiment with increasing the capacity by adding one more hidden layer to the base model. We deepen hidden layers from [1024, 512, 256, 128, 64] to [2048, 1024, 512, 256, 128, 64] for Gmail, and from [256, 128, 64] to [512, 256, 128, 64] for Drive.

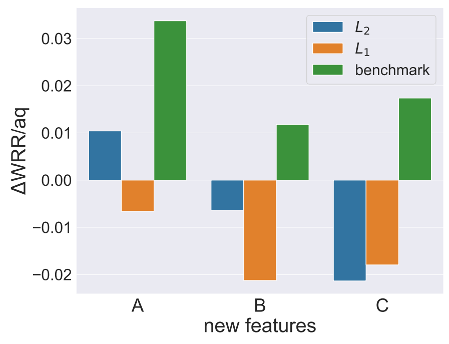
Table 1: Summary of Added New Features.

Service	Label	Comments
Gmail	A	A single feature
	B	A single feature, low coverage
	C	A group of 5 features
Drive	D	A single feature
	E	A single feature
	F	A group of 5 features

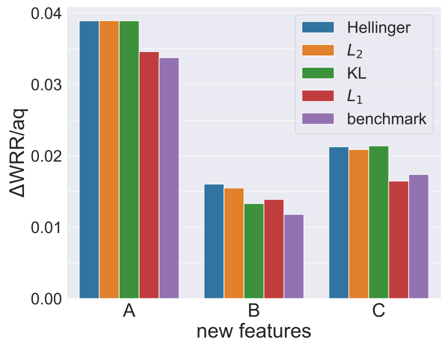
We report the results of a simple linear model for b_ϕ in additive boosting. We also tried a more complex booster such as an MLP when adding group of features but observed similar empirical performance due to the limited interaction among the newly added features. For score regularization, the regularization strength λ is tuned in $\{1, 10, 100\}$. Since the scale of the gradient is approximately proportional to λ, we set the learning rate to be inversely proportional to λ, i.e., $l = \frac{c}{\lambda}$ to facilitate stable training and c is tuned in $\{0.1, 0.5, 1.0\}$. The hyper-parameter pair (λ, l) that leads to the highest ΔWRR/aq on the validation set is selected.

6.2 Evaluation Results and Analysis

In this subsection, we report and analyze experiment results. Experiment results are summarized in Table 2. For brevity, we only present the L_2 regularizer for the listwise score regularization family. In commercial personal search systems such as Gmail and Google Drive, an overall improvement of 0.5% for WRR is typically considered significant [47].



(a) pointwise



(b) listwise

Figure 6: Comparison of pointwise and listwise score regularization for a model update that adds new features (Gmail).

6.2.1 *Pointwise v.s. Listwise Score Regularization.* We first compare pointwise and listwise score regularization. We only present the comparison of models where new features are added over the Gmail data set since we observe similar trends for other settings. From Figure 6a, we see that pointwise score regularization does not perform well. When new features are added, the ranking metric deteriorates in most cases. This result validates the concern that pointwise score regularization can be too restrictive in forcing new models to achieve a particular, specific score configuration and should be discouraged in practice. On the contrary, in Figure 6b, except for the case of applying the L_1 regularizer to feature C, the listwise score regularization approach with various regularizers all outperform the benchmark consistently.

As listwise score regularization demonstrates a clear win over pointwise score regularization, we will refer to the former as score regularization for simplicity in the remainder of this section.

6.2.2 *Adding New Features.* Empirical results for model updates that add new features are shown in Figure 7. Overall, additive boosting and score regularization with an appropriate regularizer (e.g., Hellinger, L_2 and KL) consistently outperform the benchmark model across all features. Latent cross is also quite effective in certain cases (e.g., feature E). These results validate the effectiveness of our proposed methods in general.

Additive boosting has impressive performance when the low coverage feature B for Gmail is added. From table 2, we see that compared with the benchmark, additive boosting drops the percentage of affected queries from 73.8% to 11.0% without sacrificing much ranking performance gain. Moreover, additive boosting can also be quite effective when a group of features are added (feature group C for Gmail). This is probably because the interaction between new and existing features is weak and the newly added group of features alone can achieve most of the ranking performance headroom.

For score regularization, despite the robust performance in general, Hellinger is the best performing regularizer. L_1 regularization does not perform equally well in experiments and can sometimes be worse than the benchmark (feature E). We believe this is due to L_1 being non-smooth and the propagated gradients are not as stable as that in other regularizers.

6.2.3 *Architecture Change.* As we mentioned before, only score regularization can be used to stabilize neural ranking models when their underlying model architecture changes. The performance of various regularizers are shown in Figure 8. All regularizers with appropriate regularization strength outperform the benchmark. L_2 performs the best over all four regularizers and Hellinger performs the second best.

6.2.4 *More Details on Score Regularization.* We present a more detailed comparison between score regularization and the benchmark model. Concretely, for each model update, we record the relative improve of WRR, ΔWRR%, and the percentage of affected queries $\mathbb{P}(Q_T)$ of each score regularization approach, and compare with those of the benchmark model. The difference in $\mathbb{P}(Q_T)$ is plotted on the x-axis, the difference of ΔWRR% is plotted on the y-axis, and the result is shown in Figure 9. L_2 regularization with $\lambda = 1.0$ is used to generate the plots.

Each point represents a comparison between score regularization and the benchmark model for an update and all these points have negative x-coordinates and positive y-coordinates. This means that, compared with the benchmark, score regularization not only effectively reduces the number of affected queries, but also improves the ranking metric further because it is an ensemble of the base and benchmark model, hence is better than either one.

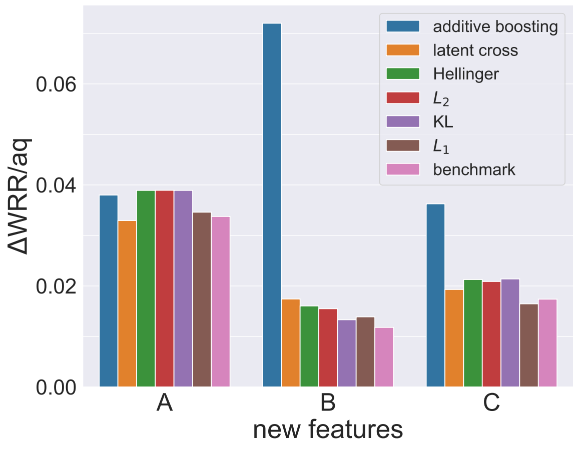
In Figure 10, we plot the performance improvement ΔWRR% and the percentage of affected queries $\mathbb{P}(Q_T)$ as regularization strength parameter λ varies. When λ increases, $\mathbb{P}(Q_T)$ decreases as expected, but ΔWRR% does not necessarily decrease, e.g., Figure 10d. Consequently, even if instability is not a concern in certain applications, one can still leverage score regularization to update a model and obtain a model that performs better than the benchmark.

Such favorable empirical behavior is due to the close connection between score regularization and ensemble methods as discussed in subsection 5.3.3, and further validates the advantage of score regularization.

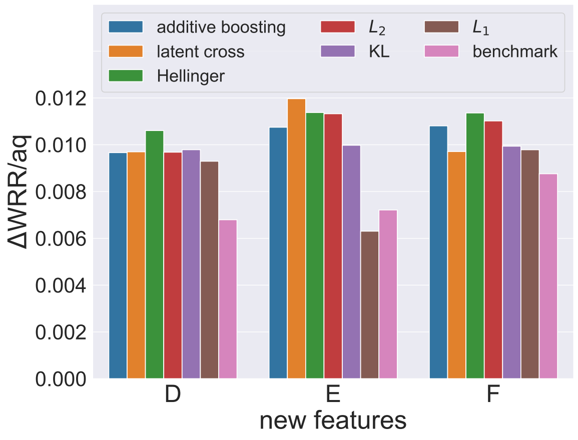
6.2.5 *Discussion of Boosting Methods.* From Table 3, we see boosting methods have considerably smaller coefficients of variation than the benchmark, also smaller than that of score regularization methods in many cases, suggesting they may be able to achieve

Table 2: Comparison of proposed methods on Gmail and Drive data sets. The units of ΔWRR% and $\mathbb{P}(Q_T)$ are percentage. The best ΔWRR/aq is highlighted in bold face for each update.

Service	Update	Benchmark			Additive Boosting			Latent Cross			L_2		
		ΔWRR/aq	ΔWRR%	$\mathbb{P}(Q_T)$	ΔWRR/aq	ΔWRR%	$\mathbb{P}(Q_T)$	ΔWRR/aq	ΔWRR%	$\mathbb{P}(Q_T)$	ΔWRR/aq	ΔWRR%	$\mathbb{P}(Q_T)$
Gmail	A	0.0338	5.34	83.0	0.0380	3.52	48.7	0.0329	4.25	67.6	**0.0389**	5.93	80.0
	B	0.0118	1.66	73.8	**0.0720**	1.51	11.0	0.0174	1.85	55.6	0.0155	2.06	69.9
	C	0.0174	2.50	75.4	**0.0363**	1.87	27.1	0.0193	2.12	57.6	0.0209	2.88	72.5
	Deeper	0.0029	0.40	73.1	—	—	—	—	—	—	**0.0066**	0.66	52.9
Drive	D	0.0068	0.55	62.4	**0.0097**	0.24	19.6	**0.0097**	0.37	29.3	**0.0097**	0.56	45.0
	E	0.0072	0.58	62.5	0.0107	0.29	20.8	**0.0120**	0.41	26.5	0.0113	0.65	44.6
	F	0.0088	0.71	63.1	0.0108	0.29	20.5	0.0097	0.50	39.9	**0.0110**	0.65	45.5
	Deeper	0.0013	0.10	59.2	—	—	—	—	—	—	**0.0067**	0.36	41.9



(a) Gmail



(b) Drive

Figure 7: Comparison of various proposed methods when adding new features.

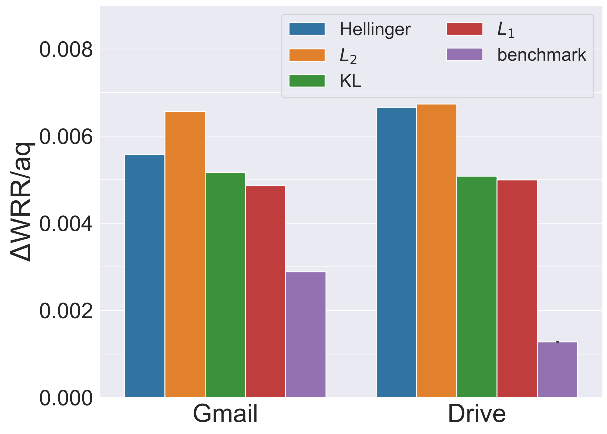


Figure 8: Comparison of score regularization with various regularizers for architecture change.

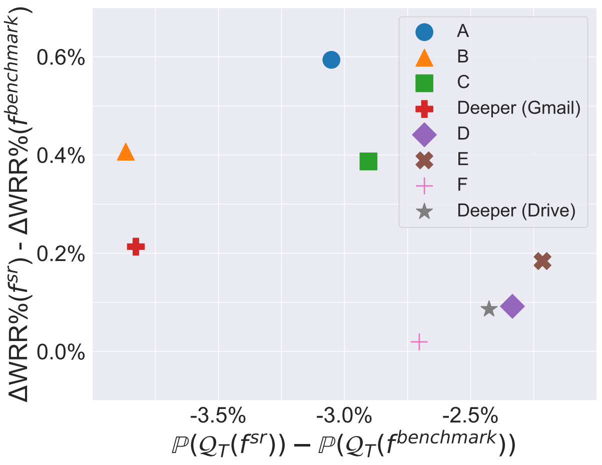
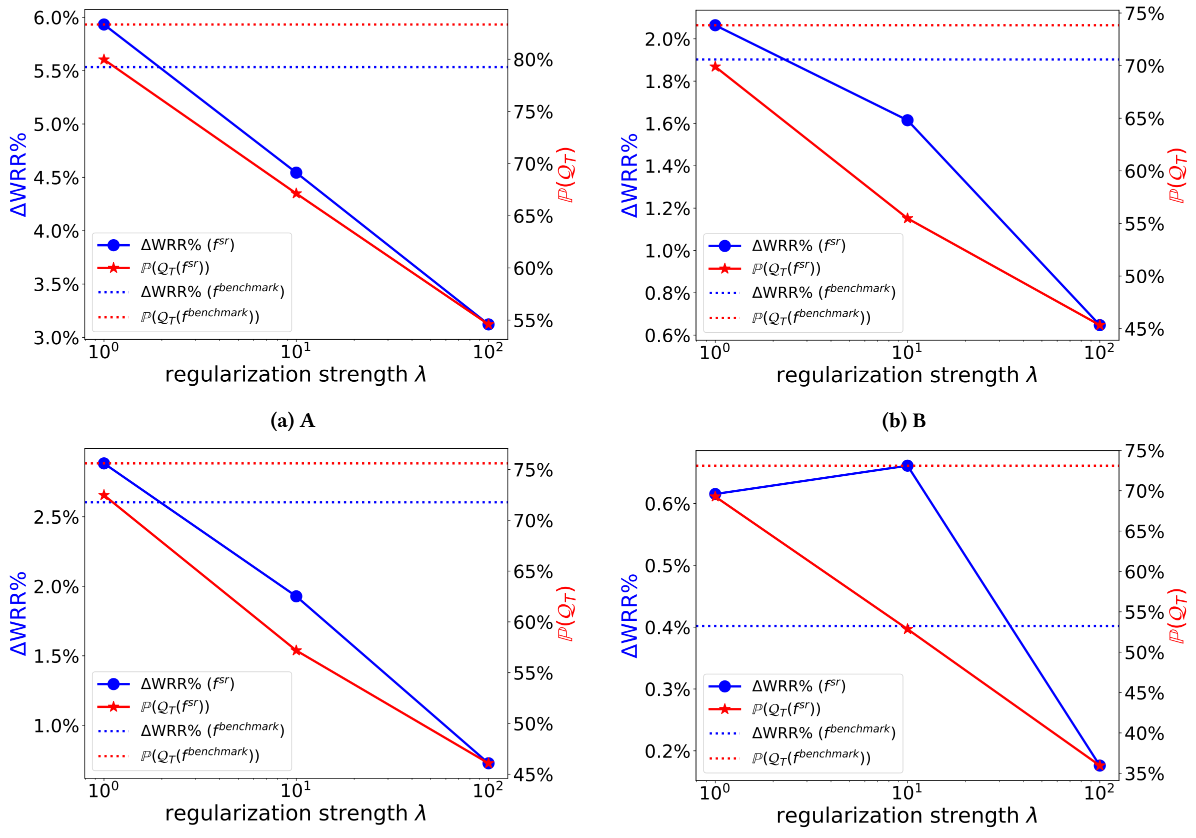


Figure 9: Detailed comparison between score regularization and benchmark for each update.

statistically significant results more efficiently in online experimentation. In practice, however, when consecutively adding new features to the base model, additive boosting and latent cross grow many branches which may be difficult to maintain. To mitigate this inconvenience and still leverage the power of boosting methods, we can either periodically retrain a deep neural model with all features or use boosting methods to identify useful new features and then launch the corresponding benchmark model.



(a) A **(b) B** **(c) C** **(d) Deeper**

Figure 10: ΔWRR% (left y-axis in blue) and $\mathbb{P}(Q_T)$ (right y-axis in red) of score regularization when regularization strength λ varies. Dotted horizontal lines represent the performance of the benchmark model. Four figures correspond to four updates in Gmail data set.

Table 3: Estimated coefficient of variation (CV) of ΔWRR/aq. The unit is 10^{-4}. The smallest CV is highlighted in boldface. $\lambda = 1.0$ for L_2 regularization.

Service	Update	Bechmark	Additive Boosting	Latent Cross	L_2
Gmail	A	70.5	54.5	**46.7**	72.7
	B	302.6	**82.6**	177.0	120.5
	C	276.3	108.7	122.8	**87.5**
Drive	D	1669.6	**749.8**	822.8	1089.9
	E	1206.9	**684.4**	859.5	928.2
	F	986.0	**662.3**	1057.6	792.8

6.2.6 Summary. Based on the above observations, we draw several major conclusions and provide guidelines for practitioners.

(1) When adding low-coverage features, additive boosting should be considered, since it is a simple yet effective approach to stabilize neural ranking models for this type of update.
(2) By observing the performance of both updates that involve adding new features and changing the underlying model architecture, we conclude that Hellinger and L_2 listwise score regularization have robust performance and should be considered as the default regularizer for score regularization.
(3) Score regularization works for both types of model updates, improves ranking quality, and suppresses the number of affected queries simultaneously. It provides a theoretically-grounded and unified approach to stabilize neural ranking models.

7 CONCLUSION

In this paper, we identified an important problem, the instability issue of neural ranking models. We discuss how this issue makes it difficult to pinpoint beneficial model updates. Additionally, unnecessary updates that change search ranking are undesirable because as users adjust from one update to another, inconsistencies increase failures in finding relevant documents [41]. As such, we proposed a new objective, improved ranking metric per affected query ΔM/aq, that can be used to balance the trade-off between quality improvements and the number of affected queries. As such, three stabilization methods were proposed, including heuristic approaches utilizing additive boosting and latent cross, as well as a theoretically motivated approach which we prove optimizes an approximation of ΔM/aq. We studied this problem in the context of personal search and empirically validated our proposed methods on both Gmail and

Google Drive search, demonstrating that our proposed methods significantly improve the stability of neural ranking model updates.

Our work opens up several interesting research directions for future work. (1) While we study the instability issue in the context of personal search, the proposed methods are generally applicable and it would be interesting to apply them to other use cases. (2) Ranking permutations below the clicked document can also be less informative when learning a model. How to stabilize neural ranking models under more sophisticated ranking comparator metrics can also be studied in the future. (3) While we focus on neural ranking models, other widely used learning-to-rank methods such as LambdaMART [6] also involve randomness in training and can suffer from instability – stabilizing these methods also warrants further investigation.

REFERENCES

[1] Tal Ben-Nun and Torsten Hoefler. 2018. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *arXiv preprint arXiv:1802.09941* (2018).

[2] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. 2018. Latent cross: Making use of context in recurrent recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 46–54.

[3] Eliot Brenner, Jun Zhao, Aliasgar Kutiyanawala, and Zheng Yan. 2018. End-to-End Neural Ranking for eCommerce Product Search: an Application of Task Models and Textual Embeddings. *Proc. of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval* (2018).

[4] Sebastian Bruch, Xuanhui Wang, Mike Bendersky, and Marc Najork. 2019. An Analysis of the Softmax Cross Entropy Loss for Learning-to-Rank with Binary Relevance. In *Proceedings of the 2019 ACM SIGIR International Conference on the Theory of Information Retrieval*.

[5] Christopher Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine learning*. 89–96.

[6] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

[7] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine learning*. 129–136.

[8] David Carmel, Guy Halawi, Liane Lewin-Eytan, Yoelle Maarek, and Ariel Raviv. 2015. Rank by time or by relevance?: Revisiting email search. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 283–292.

[9] Olivier Chapelle, Thorsten Joachims, Filip Radlinski, and Yisong Yue. 2012. Large-scale validation and analysis of interleaved search evaluation. *ACM Transactions on Information Systems (TOIS)* 30, 1 (2012), 6.

[10] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. ACM, 7–10.

[11] W Bruce Croft, Donald Metzler, and Trevor Strohman. 2010. *Search engines: Information retrieval in practice*. Vol. 520. Addison-Wesley Reading.

[12] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large scale distributed deep networks. In *Advances in neural information processing systems*. 1223–1231.

[13] Alex Deng and Victor Hu. 2015. Diluted treatment effect estimation for trigger analysis in online controlled experiments. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*. ACM, 349–358.

[14] Alex Deng, Ya Xu, Ron Kohavi, and Toby Walker. 2013. Improving the sensitivity of online controlled experiments by utilizing pre-experiment data. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*. ACM, 123–132.

[15] Werner Dinkelbach. 1967. On Nonlinear Fractional Programming. *Management Science* 13, 7 (1967), 492–498.

[16] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, Jul (2011), 2121–2159.

[17] Yoav Freund, Robert E Schapire, et al. 1996. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, Vol. 96. 148–156.

[18] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[19] Jiafeng Guo, Yixing Fan, Liang Pang, Liu Yang, Qingyao Ai, Hamed Zamani, Chen Wu, W Bruce Croft, and Xueqi Cheng. 2019. A deep look into neural ranking models for information retrieval. *arXiv preprint arXiv:1903.06902* (2019).

[20] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. ACM, 2333–2338.

[21] Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 133–142.

[22] Thorsten Joachims. 2002. Unbiased evaluation of retrieval quality using clickthrough data. (2002).

[23] Ronny Kohavi, Thomas Crook, Roger Longbotham, Brian Frasca, Randy Henne, Juan Lavista Ferres, and Tamir Melamed. 2009. Online experimentation at Microsoft. *Data Mining Case Studies* 11 (2009), 39.

[24] Ron Kohavi, Alex Deng, Brian Frasca, Toby Walker, Ya Xu, and Nils Pohlmann. 2013. Online controlled experiments at large scale. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1168–1176.

[25] Ron Kohavi, Roger Longbotham, Dan Sommerfield, and Randal M Henne. 2009. Controlled experiments on the web: survey and practical guide. *Data mining and knowledge discovery* 18, 1 (2009), 140–181.

[26] Ron Kohavi, Llew Mason, Rajesh Parekh, and Zijian Zheng. 2004. Lessons and challenges from mining retail e-commerce data. *Machine Learning* 57, 1-2 (2004), 83–113.

[27] Ronny Kohavi and Matt Round. 2004. Front line internet analytics at Amazon.com. *Santa Barbara, CA* (2004).

[28] Erich L Lehmann and Joseph P Romano. 2006. *Testing statistical hypotheses*. Springer Science & Business Media.

[29] Xiujun Li, Chenlei Guo, Wei Chu, Ye-Yi Wang, and Jude Shavlik. 2014. Deep learning powered in-session contextual ranking using clickthrough data. In *Advances in neural information processing systems*.

[30] Tie-Yan Liu et al. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009), 225–331.

[31] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[32] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[33] Bhaskar Mitra, Nick Craswell, et al. 2018. An introduction to neural information retrieval. *Foundations and Trends® in Information Retrieval* 13, 1 (2018), 1–126.

[34] Quynh Nguyen and Matthias Hein. 2017. The loss surface of deep and wide neural networks. In *Proceedings of the 34th International Conference on Machine Learning*. 2603–2612.

[35] Kezban Dilek Onal, Ye Zhang, Ismail Sengor Altingovde, Md Mustafizur Rahman, Pinar Karagoz, Alex Braylan, Brandon Dang, Heng-Lu Chang, Henna Kim, Quinten McNamara, et al. 2018. Neural information retrieval: At the end of the early years. *Information Retrieval Journal* 21, 2-3 (2018), 111–182.

[36] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[37] Rohan Ramanath, Gungor Polatkan, Liqin Xu, Harold Lee, Bo Hu, and Shan Zhou. 2018. Deploying deep ranking models for search verticals. *arXiv preprint arXiv:1806.02281* (2018).

[38] Shai Shalev-Shwartz. 2014. Selfieboost: A boosting algorithm for deep learning. *arXiv preprint arXiv:1411.3436* (2014).

[39] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*. ACM, 101–110.

[40] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web*. ACM, 373–374.

[41] Ben Shneiderman, Donald Byrd, and W Bruce Croft. 1998. Sorting out searching: A user-interface framework for text searches. *Commun. ACM* 41, 4 (1998), 95–98.

[42] Daniel Soudry and Yair Carmon. 2016. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv preprint arXiv:1605.08361* (2016).

[43] Daniel Soudry and Elad Hoffer. 2017. Exponentially vanishing sub-optimal local minima in multilayer neural networks. *arXiv preprint arXiv:1702.05777* (2017).

[44] Danny Sullivan. 2016. FAQ: All about the Google RankBrain algorithm. *Google's using a machine learning technology called RankBrain to help deliver its search results. Here's what's we know about it.[cited 2018 May 15] Available from: https://searchengineland. com/faq-all-about-the-new-google-rankbrain-algorithm-234440* (2016).

[45] Latanya Sweeney. 2002. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10, 05 (2002), 557–570.

[46] Diane Tang, Ashish Agarwal, Deirdre O'Brien, and Mike Meyer. 2010. Overlapping experiment infrastructure: More, better, faster experimentation. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 17–26.

[47] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. 2016. Learning to Rank with Selection Bias in Personal Search. In *Proc. of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 115–124.

[48] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. 2018. Position Bias Estimation for Unbiased Learning to Rank in Personal Search. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM '18)*. ACM, New York, NY, USA, 610–618. https://doi.org/10.1145/3159652.3159732

[49] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th International Conference on Machine learning*. 1192–1199.