KU LEUVEN

# Personalized Search with Deep Learning

Kshitij Goyal

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Big Data Analytics

**Thesis supervisor:**
Prof. Dr. Marie-Francine Moens

**Assessors:**
Katrien Laenen
Dr. Robin De Croon

**Mentor:**
Dr. Quynh Ngoc Thi Do

Academic year 2017 – 2018

# Preface

I would like to thank my daily supervisor Quynh Ngoc Thi Do for her constant guidance in this project. I am also grateful to my promoter, Prof. Dr. Sien Moens for providing me with an opportunity to work in this area. I am especially grateful to the authors of the original embedding paper [26] for providing us with the embedding dataset which was crucial for our work. I am also grateful to my fellow students for engaging in interesting discussions about deep learning which really helped me approach this work.

I would also like to thank my family and friends back in India for constantly providing support while I pursued my masters here in Belgium.

*Kshitij Goyal*

# Contents

# Abstract

With an ever-increasing consumption of digital information, learning to rank has become an important topic in computer science. In this thesis, we propose and develop a deep learning based learning to rank framework to deal with personalized information retrieval. There has been extensive work done on the topic of learning to rank. Because of an increasing interest in deep learning over the past few year, few works in the learning to rank have been proposed in deep learning framework. Deep learning has made significant improvement in the learning quality in some recent works. Deep networks are useful in discovering abstractions and identifying the relations between the documents and the user query. Despite the fact that user profiles are very important for personalized information retrieval, only a few of the existing works deal with the application of user profiles in the deep learning ranking models. In this thesis we apply three user profile-based learning to rank methods - Pointwise Scoring Model, Deep RankNet[4] and Margin RankNet, and evaluate the frameworks in a personalized information retrieval setting.

Our work shows that personalization of ranking using user profiles significantly improves the ranking of documents when we use the deep learning framework, with the Margin RankNet model performing better than both pointwise scoring model and Deep RankNet model in terms of precision.

# List of Figures and Tables

## List of Figures

## List of Tables

# List of Abbreviations

## Abbreviations

| | |
|---|---|
| MSE | Mean Square Error |
| MAE | Mean Absolute Error |
| NDCG | Normalized Discounted Cumulative Gain |
| CNN | Convolutional Neural Networks |
| LDA | Latent Dirichlet Allocation |
| SGD | Stochastic Gradient Descent |
| MRR | Mean Reciprocal Rank |
| MSLE | Mean Squared Log Error |

# Chapter 1

# Introduction

Information Retrieval is a process of obtaining information relevant to an information need from an information resource. Personalized Information Retrieval deals with using the user related information to improve the quality of the retrieved information in terms of the relevance. Learning to Rank is one of the fundamental problems in Information Retrieval. Learning to rank refers to the application of machine learning which deals with creating a ranking model to rank a set of documents given a query in order of relevance. Learning to rank is an essential part of problems like document retrieval (Search Engines) and collaborative filtering (Recommender Systems) and text summarization [14]. The purpose of this thesis is to study the personalized information retrieval problem by implementing multiple the learning to rank models in a deep learning framework.

The previous work on this topic is extensive. There are a number of standard algorithms which are being used for decades. In the past few years, with an increasing interest in the deep learning, few algorithms have been developed which apply deep learning frameworks to the problem of learning to rank. Deep neural networks have proven to be significantly effective in a lot of research fields in the past decade. Deep Networks are useful in discovering abstractions and learning extremely complex relations. Most of the existing algorithms include features which are based on the bag-of-words or the query-level representation. Although these works are interesting, the use of dense distributed input features in form of embedding vectors has not been explored much and is limited. In terms of data, most of the state of the art algorithms use features such as click through rate, average position etc. Also, the effect of user level embedding features on the personalization of search results is another study that has been lacking in the current literature. There is an attention based neural nets[27] method to learn the embeddings recently, which is discussed in the next chapter. Also, there are deep neural net implementations for some state-of-the-art algorithms, for example, RankNet[22], SortNet[18], ListNet[7], which are proven to be substantially better than the shallow net variants.

There have been limited research on how we can leverage just the word vectors, learned from the text data, to identify the relevance of a document in the learning to rank problem. For personalization of the information, user profiles represented as vectors can be utilized. For the purpose in this thesis, we are completely focusing on using the query and document embedding learnt from text data and the user profiles learnt using the method described in [26]. All the users, queries and documents are represented in the form of embeddings. We develop a deep learning framework including three methods for the problem, first is the simple point-wise scoring model, secondly a deep RankNet implementation using the embedding features and thirdly a pairwise deep learning model using a margin based loss function which we call Margin RankNet. We study the effect of personalization (based on user profiles) on the search results. We also study some regularization techniques and identify the best possible configuration.

This thesis is organized in following parts:

1. Chapter 2 gives a broad overview of the personalized information retrieval problem. Furthermore, background on deep neural networks and the embedding vector creation is also presented. Details on the dataset and the evaluation metrics are also provided in this chapter.

2. Chapter 3 details the methodology of all the three models.

3. Chapter 4 presents the experiments and results for each of the models with baseline comparisons.

4. Chapter 5 finishes with the conclusion and the potential future work.

# Chapter 2

# Background

In this chapter, an overview of the personalized information retrieval problem is presented. We study the existing literature of the problem along with some details on the embedding creation and artificial neural networks. We will also discuss the recent most important applications of neural network in the learning to rank problem and how our model is different from the existing algorithms. The idea is to get the reader familiar with the concepts used in the application in the later chapters.

The Chapter is divided into five sections, first section defines the Personalized Information Retrieval and provides a formal framework which is used throughout the text, Second section discusses the existing work in the field, third section provides a detailed explanation on the creation of query, document and user embeddings, fourth section presents an introduction to the artificial neural networks, fifth section details the evaluation metrics and software tools used for the experimentation in the next chapters.

## 2.1 Personalized Information Retrieval

Information Retrieval is the process of obtaining information relevant to a query from a source of information. Personalization aims at improving the quality of the retrieval process by taking into account the information of individual users. Even though user preferences can be complex and heterogeneous, these user profiles can be learnt by studying the historical behaviour of users. These profiles can be then utilized in the learning to rank models to improve the retrieval quality.

### 2.1.1 Learning to Rank

Learning to rank is an application of machine learning which deals with creating a ranking model for an information retrieval system. There are many applications of learning to rank such as document retrieval, collaborative filtering, sentiment analysis and text summarization [14]. The objective of a learning to rank model is to figure out the relative ranking from a list of documents in order of relevance. In other words, ranking the document with higher relevance higher than the document with low relevance. Understanding the degree of relevance between the document and the query is a complicated task and the classical machine learning algorithms under-perform in this application. Hence, there has been an extensive research on the topic with numerous proposed algorithms.

### 2.1.2 Framework

In our work, we are dealing with a supervised learning approach, which means that we create a ranking model given a training dataset and use this model to predict (determine) the ranking of a new set of documents for a given query. In addition to that, we are using the personalized user features which are represented in form of embeddings. Training data consists of user-query pairs $u_i$ and $q_j$ for a user $i$ and their associated documents $x_k^{(i,j)}$. Furthermore, each of the documents for a user-query pair is associated with a relevance or implausibility score (multiple scenarios for training are considered which are explained in later chapters). Using the training data, the model $h$ is learnt to predict these scores for each of the documents given a user and a query.

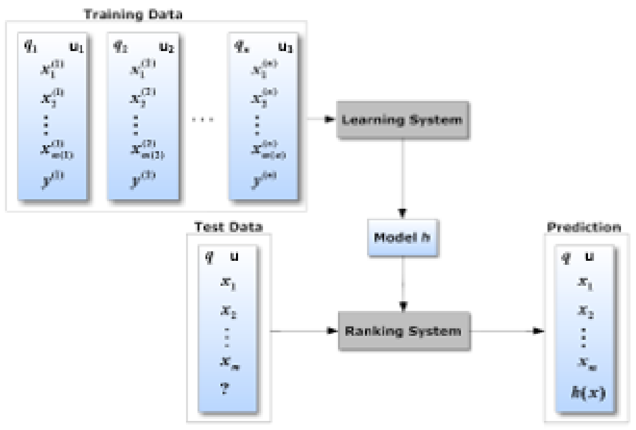

Figure 2.1: Framework[14]

In the literature, the features that define a document or a query can be of three types. Query independent features, dynamic features which depend both on query and document and query features which only depend on the query. However, in our work, we are using only the learnt embeddings as features[1]. Further, the relevance of the documents are presented in multiple different ways:

1. Multiple levels of relevance. It could be binary ("good","bad") or multiple categories ("excellent", "good", "bad", "horrible").

[1] https://en.wikipedia.org/wiki/Learning_to_rank#Feature_vectors

2. A complete ordered list of documents in order of relevance

3. Pairwise relevance is established for any two documents from the set

Following [26], we are considering a binary level of relevance, only one document for each user-query pair is known to be relevant, all other documents are considered as non-relevant. Using this binary relevance, pairwise relevance is created for documents and used in our RankNet implementation and the Margin RankNet model.

## 2.2 Literature Review

Personalized information retrieval remains an interesting topic and a lot of research has been done in the past two decades. With the recent developments in neural networks, there has been some breakthroughs in the field. We present some of the state of the art algorithms in this section. Before moving to the algorithms, it's important to point out different approaches that are being used for the learning to rank problem.

As explained in [14], there are three main approaches: *Pointwise*, *Pairwise* and *Listwise*. In pointwise approach, each pair of query-document has a numerical score, reflecting the relevance ranking, and the ranking problem can be reduced to a classification or regression problem. In the pairwise approach, two documents for a given query are compared and the more relevant document is identified. The pairwise approach can be seen as a binary classification problem. In listwise approach, an objective function is minimized by considering a list of documents and queries. Following are a few state of the art algorithms which are being used in the literature.

**Regression or Classification:** In a pointwise framework, each document for a query is attached with a relevance score. A lot of existing machine learning algorithms can be applied to learn that score and provide a ranking of the documents based on the relevance score. Training data could have multiple levels of ordinal values instead of just a binary score (relevant/irrelevant). Algorithms like linear regression, decision trees[17] can be easily applied for this. In one of the algorithms that we propose, we use a deep learning framework and we learn these scores in a pointwise learning manner. We have used the root mean squared error as our loss function.

**PRank[9]:** The PRank algorithm uses a pointwise approach to minimizes the ranking loss. The algorithm considers instances $x_1$ and $x_2$ whose rank are from a finite set $Y = (1, 2, ....k)$, $x_1$ is more relevant than $x_2$ if $y_1 > y_2$. The objective is to learn a ranking rule $H : R^n \rightarrow Y$ which maps an instance to a rank. The algorithm defines a vector $w$ and a set of k thresholds $b_1 \leq b_2 \leq .... \leq b_k = \infty$. For computing the rank of an instance $x$, the dot product $w.x$ is computed and the rank of the instance is assigned to be the index of the smallest threshold $b_r$ such that $w.x < b_r$. Iteratively the algorithms calculates the rank of each training instance and modifies $w$ and $b$ to minimize the classification error.

**RankNet[4]:** RankNet uses a pairwise learning approach to model the ranking function. RankNet uses a neural network framework to learn a score function $f(x)$

for an input feature vector $x \in R^n$. For a query $q$, two documents $x_i$ and $x_j$ of differing relevance are chosen. The model then computes the relevance scores for each of the documents as:

$$s_i = f(x_i); s_j = f(x_j) \tag{2.1}$$

If we know that the document $x_i$ is more relevant than $x_j$, the probability of document $i$ being ranked higher than document $j$ is calculated as:

$$P_{ij} = \frac{1}{1 + e^{-\sigma(s_i - s_j)}} \tag{2.2}$$

Where $\sigma$ describes the shape of the sigmoid. Then the cross entropy cost function is applied to penalize the deviation of the output probabilities from the correct probabilities $\overline{P_{ij}}$

$$C = -\overline{P_{ij}} log P_{ij} - (1 - \overline{P_{ij}}) log(1 - P_{ij}) \tag{2.3}$$

Where $\overline{P_{ij}} = \frac{1}{2}(1 + S_{ij})$ and $S_{ij} \in (0, 1, -1)$. $S_{ij}$ is 1 if document $i$ is more relevant than document $j$, -1 is document $j$ is more relevant than document $i$ and 0 if they have the same label. Substituting $\overline{P_{ij}}$ in equation (2.3) and combining it with equation (2.2), gives the following expression of cross entropy function to be minimized in training:

$$C = \frac{1}{2}(1 - S_{ij})\sigma(s_i - s_j) + log(1 + e^{-\sigma(s_i - s_j)}) \tag{2.4}$$

This cross entropy function is then minimized using Stochastic Gradient Descent[12]. The original paper applies RankNet to a shallow network. In [22], a personalized deep RankNet model is presented, the model is first trained in a deep learning framework with user-independent data. Then the model is adapted by training individual models based on user's search history. In our work we have extended the RankNet implementation to a deep network with embedding based features using batch learning as discussed in [5]. It is to be noted that instead of a neural net, boosting algorithms can be used for optimizing the loss[5].

**LambdaRank[6]:** LambdaRank improved upon the RankNet algorithm by using a different loss function. In the training of RankNet as the gradient descnet step, only the gradients are required and not the cost itself. It has been identified in the [5] and [6] that leveraging the changes in NDCG in the gradient calculations improves the results significantly. The gradients are interpreted as the required changes to convert a ranking to the correct ranking of documents and NDCG gives a good approximation of that.

**LambdaMART[5]:** LambdaMart is a boosting tree algorithm implementation of LambdaRank. LambdaMart uses gradient boosting decision trees using the cost function derivation of LambdaRank. In general LambdaMART tend to perform better than RankNet and LambdaRank.

**SortNet[18]:** SortNet uses a shallow neural network architecture to identify the relevance between two objects $x_1$ and $x_2$. The algorithm uses a feed-forward network along with a constrained weight architecture to ensure the equivalence relations are satisfied. It identifies the probabilities of both $x_1$ and $x_2$ being more relevant. These pairwise preferences are then combined by a global sorting function to produce a sorted list of documents. It's an iterative procedure and after each iteration, the algorithms focuses on the misclassified pairs. The training stops when there's no significant change in the learning set in consecutive iterations.

**ListNet**[7]**:** ListNet is a listwise approach to estimate the scores of a list of documents. Given a list of documents $X_i = (x_{i1}, x_{i2}, ..., x_{in})$ and an original score list $Y_i = (y_{i1}, y_{i2}, ..., y_{in})$, the function will estimate the scores $\overline{Y_i} = (\overline{y_{i1}}, \overline{y_{i2}}, ..., \overline{y_{in}})$. The training is done by optimizing a loss function of the form $sum_{i=1}^{m} L(Y_i, \overline{Y_i})$, where $m$ is the number of lists in the training data. $L$ is the listwise cost function which is proposed to be a cross entropy function between the probability distributions on permutations of these two score lists. This algorithms has a very high time complexity.

**Attention Based Deep-Net**[27]**:** This is a listwise learning to rank approach which proposes an attention-based neural network mechanism to better incorporate the embeddings of queries and related documents. The authors propose that instead of using just one model to learn embeddings, a committee of different models improve the performance of the learnt embeddings. This is argued to be true for both image (using CNN[12]) and text vectorization (using tf-idf, LDA[3], etc.) problems. The paper proposes a Recurrent-Neural Network architecture to learn different embeddings of the queries and documents over time based on changing weights. The paper also discuss a decoder mechanism to learn the rank of the documents in a listwise fashion, which is based on similarity between the query vector and the document vector.

**Other Learning to Rank Algorithms using Deep Neural Nets:** In the recent years there has been a number of algorithms which implement a deep learning framework in the learning to rank problem. In [21], Convolutional Neural Networks[12] are utilized to extract features from the raw query-document pairs before a pairwise learning is performed using a deep RankNet algorithm. In [15], the query centric contexts are extracted to figure out the position of the contexts in the documents. This is then utilized for the final global ranking model using Recurrent Neural Networks. In [20], Convolutional Neural Nets are again used to converts texts to vectors, which are then utilized to infer the similarity between queries and documents. In [8], A weight sharing technique is utilized to learn preference between two documents for a given query.

**Search Personalization with Embeddings**[26]**:** This algorithm provides an approach for utilizing the text data in the queries and documents to learn profiles of users in form of vector embeddings and then using these learnt profiles directly to rank the documents in the order of relevance. An implausibility score is estimated along with user profile creation to learn the ranking of documents. Stochastic Gradient Descent[12] is used to minimize a margin based loss function. The work we present in the thesis is an extension of the work done in this paper. We separate the vector embedding and the ranking step. In our work, deep learning is used to utilize the learnt embeddings and user profiles to rank the documents for a query. Formal details are provided on the embedding creation for query, document and user in the next section of this chapter. Multiple deep learning architectures are discussed in the next chapter. It is to be noted that we have not done any work on the embedding creation, we requested the embedding data already created from the authors of [26]. Our work mostly focuses on utilizing the embeddings to create ranking models.

## 2.3 Text Embeddings

Before moving to the methodology, it is important to elaborate on the type of data we would be using. Classical learning to rank models deal with three types of features[14]. Firstly, static features, which mainly depend upon the documents only. Second type of features are dynamic features, which depend upon both document and query. Third types of features are only dependent on the query. For the purpose of this thesis, instead of calculating these features the algorithm completely depends upon the embedding representation all of users, documents and queries. Query logs from a commercial search engine are used as the base data for the embedding modelling. Firstly, document embeddings are modelled by using $k$ topics extracted from relevant documents. Latent Dirichlet Allocation (LDA)[3] is used to automatically learn k topics from the documents. After LDA model is trained to find out the topic distribution of each distribution, topic proportion for each document is used as it's embedding as explained in [26]. Query embeddings are created in a similar manner[26]. Following subsections detail the process of generating these embeddings from the query and document text data. Before moving to the embedding creation, LDA is briefly described as well. It is to be noted that we have not performed these calculations ourselves as we couldn't get hold of the raw text and the query log data. Instead we received these embedding data from the authors of [26] who proposed this method originally. The focus of our work is how to utilize the vector embeddings for queries, users and documents as features to create learning to rank models in a deep learning framework.

### 2.3.1 Latent Dirichlet Allocation (LDA)

Topic modelling[13] is a statistical technique to discover the abstract topics that occur in a collection of documents. The number of topics $k$ is defined a-priori. For a collection of documents $D = d_1, ...., d_n$, collection of topics $Z = z_1, ...., z_k$ and number of words $W = w_1, ...., w_m$, topic models try to estimate the probabilities $P(z_k|dj)$ and $P(w_i|z_k)$. These probabilities represent the distribution of topics in a document and distribution of words in a topic respectively. LDA[3] is a generative probabilistic topic model. In LDA, priors $\alpha$ and $\beta$ are assumed for the topic and word distributions respectively. Figure 2.2 gives a graphical depiction of an LDA model. As Explained in [3], following generative mechanism is assumed:

1. Sample k times $\phi \sim Dirichlet(\beta)$

2. For each document $d_j$, sample $\theta \sim Dirichlet(\alpha)$
3. For each work position $i \in d_j$, sample $z_{ji} \sim Multinomial(\theta)$ and $w_{ji} \sim Multinomial(\phi, z_{ji})$

The probability distributions are learnt using the iterative process of Markov Chain Monte Carlo (MCMC) sampling[25].

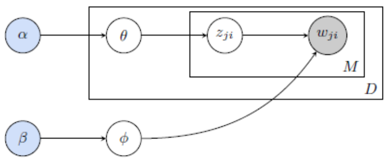

FIGURE 2.2: Latent Dirichlet Allocation[3]

### 2.3.2 Document Embeddings

For a document $d$, let the document embedding is $v_d$. LDA is used to learn $k$ topics from the relevant documents as explained in [26]. This LDA model gives us the probability distribution of topics in each document. This topic proportion vector is then used as an embedding vector for each document. More formally, for the $z^{th}$ element of $v_d$, $v_{d,z} = P(z|d)$ where $P(z|d)$ is the probability of the topic $z$ given the document $d$. The length of each document embedding vector is equal to $k$, the number of topics learnt using LDA.

### 2.3.3 Query Embeddings

As Explained in [26], embedding vector for a query q, $v_q$ is also represented using the topic modelling. $z^{th}$ element of the vector embedding for a query $q$, $v_{q,z} = P(z|q)$, where $P(z|q)$ is the probability of the topic $z$ given query $q$. $P(z|q)$ is then calculated as the mixture of topic probabilities of topic $z$ given the document relevant to query $q$. For a set of $n$ relevant documents $D_q = (d_1, d_2, ..., d_n)$, the $P(z|q)$ is defined as:

$$P(z|q) = \sum_{i=1}^{n} \lambda_i P(z|d_i) \tag{2.5}$$

where $\lambda_i$ is the exponential decay function of document $d_i$(having rank $i$ in $D_q$) and is defined as:

$$\lambda_i = \frac{\delta^{i-1}}{\sum_{j=1}^{n} \delta^{j-1}} \tag{2.6}$$

$\delta$ is the decay hyper parameter[26] $(0 < \delta < 1)$. This decay function makes sure that the document with higher relevance(lower rank) for a query gets higher weight in the topic probability for the vector embedding of that query.

### 2.3.4 User Embeddings

The third type of embedding data created in [26] is the user embedding. User embeddings ($v_u$) are learnt as a part of the ranking model training. This model is the baseline for the models discussed in the next chapter. Following [26], the goal is to learn a score function $f$ which calculates the implausibility value for a triple $(u, q, d)$. For a user $u$ and query $q$, the implausibility value for a correct (more relevant) triple $(u, q, d)$ is smaller than an incorrect (less relevant) triple $(u, q, d^{'})$. The score function is assumed to be of the following form:

$$f(u, q, d) = ||W_{u,1}v_q + v_u - W_{u,2}v_d||_{l_1} \tag{2.7}$$

Here matrices $W_{u,1}$ and $W_{u,2}$ are the model parameters along with $v_u$ which represents the user embeddings. $v_q$ and $v_d$ are the query and document embeddings learnt using LDA[3]. To learn the parameters($W_{u,1}$, $W_{u,2}$ and $v_u$), a margin based objective function is minimized using Stochastic Gradient Descent[12].

$$L = \sum_{(u,q,d)\in G;(u,q^{'},d^{'})\in G^{'}} max(0, \gamma + f(u, q, d) - f(u, q^{'}, d^{'}) \tag{2.8}$$

where $\gamma$ is the margin hyper-parameter, $G$ is the set of correct triples and $G^{'}$ is the set of incorrect triples created by corrupting the correct triples by either changing the relevant documents to irrelevant documents for a query-user pair or by changing a relevant query-document pair to an irrelevant one for a given user. This loss function is then minimized with the constraints that $||v_u||_2 \leq 1$, $||W_{u,1}v_q|| \leq 1$ and $||W_{u,2}v_d|| \leq 1$ to learn the matrices and the user embeddings. A simpler model is also learnt by fixing the matrices $W_{u,1}$, $W_{u,2}$ as identity matrices. Both of these models are baselines for our proposed methods.

The user profile data that we used in our work was created by initializing the weights randomly. These user embeddings capture the user's topical interests and could be used as features in any learning to rank model. It's detailed in the next chapter how a similar margin loss function can be used to find the ranking of documents with user embeddings as part of the input.

## 2.4 Artificial Neural Networks

Artificial Neural Networks[12] are computing systems which are inspired by the structure of human brains in the sense that there are intermediate nodes and neurons coming in and out of these nodes. These neurons propagate information forward(or backwards). In mathematical terms, these neurons have certain weights which are learnt as part of the training. Figure 2.4 depicts the simplest variant called Perceptron. Perceptrons are binary classifiers and just consist of one hidden layer. Perceptrons are useful for modelling binary classification problems but don't perform all that well with complex modelling settings. Hence, multi layer networks are used. An artificial neural network can be characterized by following components

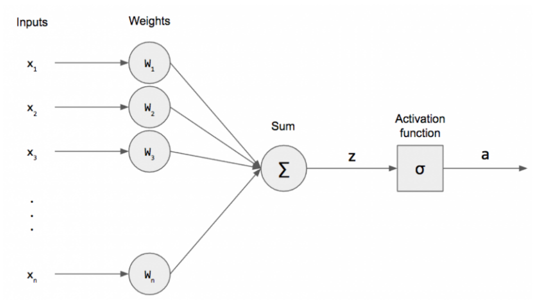

FIGURE 2.3: Single Layer Perceptron[2]

**Neurons:** neurons are the basic unit of a neural network. A neuron has potentially multiple inputs, an activation process and one output. Each input has an assigned weight and a bias term, the neuron accumulates all the inputs as a weighted sum, applies an activation function on the collection and produces an output to the next neuron (if there is one). Every neuron has an assigned activation function. Figure 2.4 depicts a model of a neuron. The output $y$ is the input for the next neuron and $f$ is the activation function.

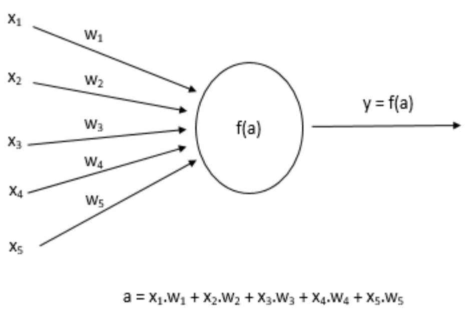

FIGURE 2.4: A Model of a neuron

**Activation Function:** Activation function is the function associated with every neuron in the network. Referring to figure 2.4, activation function $f$ is applied to the weighted sum $a$ and the output $y$ is propagated to the next neuron. Activation functions are a very important part of the neural net because they add the non-linearity to the model which is very effective in learning complex abstractions in the data[24]. A neural net with all linear function is theoretically similar to a linear regression model. Through the inception of the neural nets, a number of activation functions have been suggested. Sigmoid and tanh are the non-linear function with output values in range $[0, 1]$ and $[-1, 1]$ respectively. Rectified linear unit (ReLU), $f(x) = max(0, x)$, is the most widely used activation function in the neural nets

[2]https://pythonmachinelearning.pro/perceptrons-the-first-neural-networks/

because it removes the problem of vanishing gradient[12]. Figure 2.5 depicts the three activation functions mentioned above.

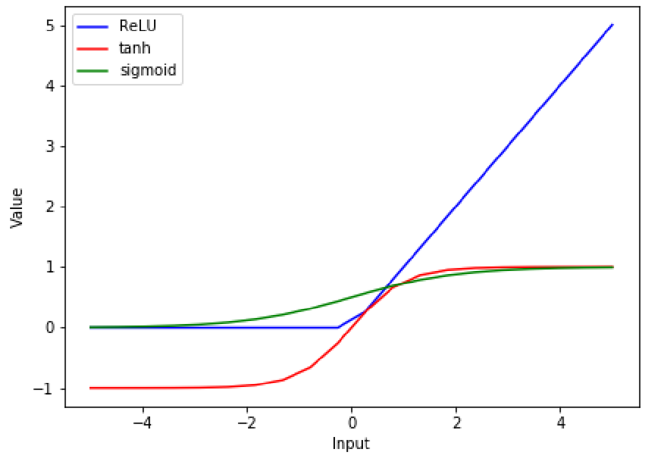

Figure 2.5: Activation Functions

**Learning Rule:** Learning rule or algorithm is another important factor that describes a neural network learning. Every algorithm or rule aims to learn the final wights of the connections between the neurons in the network. These weights define the relationship between the inputs and the outputs. The learning algorithm is something that dictates how these weights are learned. A learning rule comprises of the input and output structures, the process of weight updates and the activation functions. All these are highly dependent upon the application and the user. Although there are some standard non-linear functions and updating methods, there is no general learning framework. The architecture of the network (number of layers, level of connectivity, activation function) and the weight update process is highly practice-driven and problem dependant.

### 2.4.1 Feed-forward Neural Networks

Feed-forward neural networks are a family of neural nets where the output of one layer is always propagated forward, i.e., there's no loop in the architecture, hence they are called "feed-forward". Feed-forward networks are multi-layer networks containing one input layer, one or more hidden layers and an output layer with some neurons in each layer. Figure 2.7 shows an example of a 2 hidden layer feed-forward network.

In most cases, there are non-linear functions assigned to each of the neurons. These function help learn the non-linear abstractions between the inputs and outputs. As mentioned before, the weights assigned to knowledge transfer between two neurons

[3] https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a

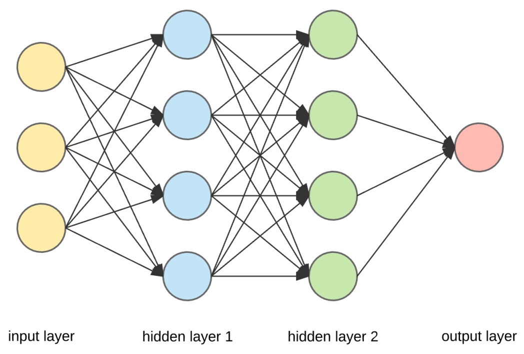

FIGURE 2.6: Feed Forward Neural Network[3]

define the relationship between inputs and the outputs. Given some training data, these weights are learnt through a training process

**Training:** To learn the weights between neurons and layers, the training algorithms that is used is called backpropogation[12]. Backpropogation is a method to adjust the interconnecting weights between layers to reduce the prediction error. In neural network training, backpropogation is generally used by the gradient descent algorithm which calculates the gradient of a loss function with respect to the weights and then adjusts the weights in the negative direction of the gradient. This minimizes the loss function with respect to each weight. The choice of the loss function is very much application dependant. For classification and regression problems, mean squared error (MSE) or mean absolute error (MAE) are generally used. But the choice of loss function is not at all limited to these. This is an interesting area of deep learning research.

Gradient descent[12] is an iterative process which tries to adjust the weights so as to minimize the loss function as much as possible. Figure 2.7 provides an illustration of the SGD process graphically.

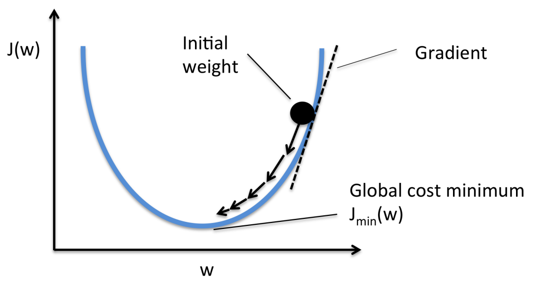

FIGURE 2.7: Stochastic Gradient Descent[4]

It can be described as follows:

1. All Weights are initialized randomly.

[4]https://www.quora.com/What-is-Stochastic-Gradient-Descent

2. For a training instance, the output is calculated by using the forward propagation in the network.

3. Loss is calculated based on the desired output and the calculated output.

4. Gradient of the loss is computed with respect to each of the weight variables.

5. Each weight is adjusted in the negative direction of the gradient with a learning rate multiplier.

This process is called Stochastic Gradient Descent (SGD). The process is stopped when the there's no improvement in the loss. The learning rate is an important variable in the learning process. If the learning rate is high, the learning process is quick but not accurate enough. On the other hand, if the learning rate is small, the learning is very slow but much accurate. Hence, it's up to the user to select a desirable learning rate practically.

The process explained above is a very general form of learning. The update can be done in a batch mode (where the update doesn't happen with every new training instance but on a batch of instances) or it could happen in a pairwise manner (like in RankNet[4]) or once for every epoch (it is discussed further in the next chapter). Also, variants of SGD - AdaGrad[10], RMSProp[19], Adam and Kalman-based Stochastic Gradient Descent (kSGD)[16] are also in use which deal with issues of learning rate selection and non-convex loss optimization.

**Validation:** One of the major problems in any of the machine learning algorithm is the problem of over-fitting[12]. Over-fitting happens when the model becomes very specific to the training data and doesn't generalizes well on new data sets. To avoid that, a validation dataset is used in the training process. After every update step, the loss is calculated on the validation set. When over-fitting happens, the validation loss starts to increase instead of decreasing. The training should stop at that instance. There are other ways to avoid over-fitting such as regularization. Validation test is also very useful in parameter tuning. To identify the ideal user dependant parameters, validation set can be used to fine-tune the configuration.

**Dropout:** Dropout is a regularization technique[23] for reducing over-fitting in the neural networks. In a neural network, a dropout mechanism is used as the regularization measure where a certain number of nodes are dropped from some of the layers. This process makes the neurons robust and they become less dependent of the neighbouring neurons. Neurons are dropped randomly in a probabilistic manner with a probability $p$. This probability is set by the user and is an important part of the model creation.

### 2.4.2 Deep Neural Networks

A Deep Neural Network[12] architecture consists of multiple interconnected hidden layers. Deep networks can model complex non-linear relationships. The high number of layers enables the model to learn very low level abstractions in the data. Deep learning is very widely used in the filed of image recognition. Deep Neural Nets can also be used in an unsupervised learning framework which has a number of applications in the field of computer vision, speech recognition etc.

## 2.5 Evaluation Metrics and Software Tools

### 2.5.1 Evaluation Metrics

In this part, we present the evaluation measures that we have used in our work. The dataset we use only contains one relevant document for each of the query, the evaluation measures used are based on just the most relevant document. For this purpose, we have evaluated the performances of our models with Precision and Mean Reciprocal Rank (MRR).

**Precision:** Precision measures the fraction of identified relevant documents which are actually relevant. Precision represents the probability of a retrieved document being relevant. Precision is a measure of quality of ranking in terms of exactness.

$$P@1 = \frac{\text{Number of Truly Relevant Documents in the identified Relevant documents}}{\text{Total Number of Identified Relevant Documents}} \tag{2.9}$$

**Mean Reciprocal Rank (MRR):** Reciprocal Rank of a query is the multiplicative inverse of the first (here only) relevant document. For example, if for a query, the calculated rank of the relevant document is 3, the reciprocal rank would be 1/3. Mean reciprocal rank is the average of the reciprocal ranks over all queries. MRR represents the quality of ranking in terms of how far a relevant document is ranked from the top. For a set of queries $Q$:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{Rank_i} \tag{2.10}$$

### 2.5.2 Software Tools

For implementation of the models described in the next chapter, we have used the following libraries in python:

**Keras[5]:** It is an open source neural network library written in python. It is capable on running on top of Tensorflow or Theano. Keras adds a lot of functionality over the base libraries and provides a very user-friendly programming framework.

**Chainer[6]:** Chainer is an open source deep learning written in python. It works over Numpy library. This provides an intuitive mechanism for the customized loss function.

[5] https://keras.io/
[6] https://docs.chainer.org/en/stable/

# Chapter 3

# Methodology

The aim of the thesis is to propose and implement deep learning based learning to rank models for the personalized information retrieval problem. We aim to utilize the embeddings, created using the text data, as features and study the effect of these features on the performance of the models. Also, we aim to personalize the ranking using the user profiles which are learnt using the embedding model [26].

In this chapter, we present the terminology and different methodologies that we applied for the learning to rank problem. First is the simple pointwise scoring model where the output is a relevance score and the documents are ranked based on this score. Secondly, the deep net implementation of the existing RankNet[4] algorithm, here we also learn a relevance score function through the pairwise learning approach. We then propose a pairwise learning method where we use a margin base loss function to learn an implausibility value function, we call this model Margin RankNet. For a query, the document having lower implausibility value will be ranked higher. All these three models have a similar architecture - input layer, multiple hidden layers and single output neuron. The models differ in the learning rule and the output function. The architecture is best depicted in figure 3.1.

Our work is in extension to the original paper [26], the methods applied in the paper are considered as the baselines. The Search Engine rank is another baseline that we have considered. The comparison of our models with the baselines is also detailed in the next chapter along with the experimentation results. The dataset we have used in our work is provided to us by the authors of [26]. The processed data contains the vector representation of the queries, documents and user profiles. Apart from this, we use the search engine data which contains search engine ranks of each document and the relevant document corresponding to a user-query pair. We have a training set which is used for model creation, a development set which is used for parameter tuning and a test set which is used for baseline comparison.

## 3.1 Terminology

In this subsection, the terminologies used for this thesis are discussed. Let $Q$ denotes the set of queries, $U$ denotes the set of users and $D$ denotes the set of documents.

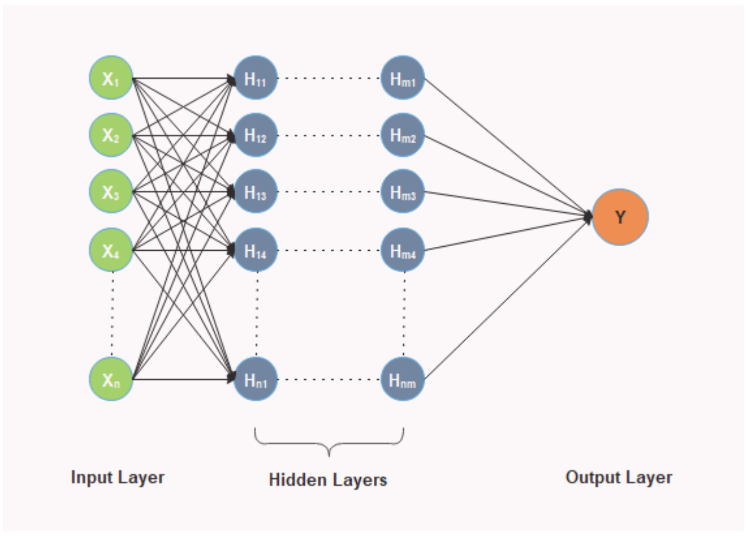

Figure 3.1: Deep Learning Architecture for Personalized Search

A triple of (user, query, document) is represented as $(u, q, d)$ where $u \in U$, $q \in Q$ and $d \in D$. Each of the user-query pair $(u, q)$ is associated with a set of documents $D_{(u,q)} = (d_1, d_2, ....d_n)$, the $n$ here is not fixed and could vary from pair to pair. We consider a binary representation in our work as we just have one relevant document for each user-query pair in the dataset.

Each of these documents has a real valued non-negative score. $Y_{(u,q)} = (y_1, y_2, ....y_n)$ represents the scores for each of the documents related to the user-query pair $(u, q)$. Although these scores are learnt from the models, these are not necessarily required for the training in a few of the discussed models. Our RankNet and Margin RankNet just require the pairwise relevance for the training, this is discussed in the more detail in the later chapters. The non-negative scores learnt from the model are used to find the relevance order of the documents in the testing set. In terms of features, each of the users, queries and documents are represented by vector embeddings $v_u$, $v_q$ and $v_d \in R^k$ respectively. Document and query embedding are learnt using topic modelling [26][3] and the user embeddings are learnt by the method described in [26].

## 3.2 Point-wise Scoring Model

In this model, we propose a point-wise scoring model which will learn a scoring function $f$ which calculates a relevance score for a triple $(u, q, d)$. Given a user and a query, a set of document D is returned from the search engine. The scoring function calculates the relevance score for each document and the documents are ranked in order of the relevance score (document with higher relevance score is ranked higher). The loss function optimization happens using the stochastic gradient descent[12]. Since we don't have any relevance score in the dataset, we manually assigned the scores to the documents. We have one relevant document for a query $q$, we give this document a score of 1. All other documents are assigned a score of 0. Hence, the input to the network is a triple $(u, q, d)$ which is represented by a vector of size 600 (concatenation of three 200 sized vectors), with the expected output either 1 or 0 (depending upon the relevance).

Although in this section we explain the model in terms of Mean Squared Error, in the experiments part of the thesis we try following loss functions[1]:

1. Mean Squared Error (MSE) = $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$
2. Mean Absolute Error (MAE) = $\frac{1}{n}\sum_{i=1}^{n}|(y_i - \hat{y_i})|$
3. Mean Squared Log Error (MSLE) = $\frac{1}{n}\sum_{i=1}^{n}(log(y_i + 1) - log(\hat{y_i} + 1))^2$
4. Log-Cosh Loss = $\frac{1}{n}\sum_{i=1}^{n} log(cosh(y_i - \hat{y_i}))$

Here $n$ is the number of instances, $y_i$ is the predicted value and $\hat{y_i}$ is the true value. With MSE, we now present the optimization of the loss function. For a given user $u$ and query $q$, the model calculates the relevance score $s_i$ for a document $d_i$ as

$$s_i = f(u, q, d_i) \tag{3.1}$$

For a batch size of $n$, the loss can be written as:

$$L = \frac{1}{n}\sum_{i=1}^{n}(s_i - \hat{y_i})^2 \tag{3.2}$$

Where $\hat{y_i}$ is the actual score of the document $d_i$. For a weight $w_k$, the derivative of the loss is:

$$\frac{\partial L}{\partial w_k} = \frac{2}{n}\sum_{i=1}^{n}(s_i - \hat{y_i})\frac{\partial s_i}{\partial w_k} \tag{3.3}$$

Also, the formula for the weight updates according to the Stochastic Gradient Descent[12] is:

$$w_k \to w_k - \eta\frac{\partial L}{\partial w_k} \tag{3.4}$$

With equation 3.3 and 3.4,

$$w_k \to w_k - \eta\frac{2}{n}\sum_{i=1}^{n}(s_i - \hat{y_i})\frac{\partial s_i}{\partial w_k} \tag{3.5}$$

Equation 3.5 represents the update formula for each of the weights. $s_i$ depends on the network architecture. The update happens until there improvement in the training and validation loss. We have trained this model for 10 epochs. Experimentation on the network architecture and loss function as well as regularization is presented in the next chapter. We also compare the model with the baselines in the next chapter.

[1] https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0

## 3.3 Deep RankNet

In this second model, we implement the existing RankNet[4] model with a deep architecture similar to [22]. In our model we train the personalized global RankNet model with user-dependent inputs (user embeddings) unlike in [22], where they do the personalization as a second step. RankNet uses a pairwise learning approach to model the ranking function. For an input feature vector $x \in R^n$, where $x$ is a represented by a triple $u_i, q_i, d_i$, a scoring function $f$ is learnt. Following [4], given a user $u$ , query $q$ and two documents $d_i$ and $d_j$ differing in relevance, the model learns the relevance scores for each of the documents as:

$$s_i = f(u, q, d_i); s_j = f(u, q, d_j) \tag{3.6}$$

If the document $d_i$ is more relevant than $d_j$, then the probability of document $d_i$ being ranked higher than $d_j$ is calculated as:

$$P_{ij} = \frac{1}{1 + e^{-\sigma(s_i - s_j)}} \tag{3.7}$$

Where $\sigma$ describes the shape of the sigmoid. Then the cross entropy cost function is applied to penalize the deviation of the output probabilities from the correct probabilities $\overline{P_{ij}}$

$$C = -\overline{P_{ij}} log P_{ij} - (1 - \overline{P_{ij}}) log(1 - P_{ij}) \tag{3.8}$$

Where $\overline{P_{ij}} = \frac{1}{2}(1 + S_{ij})$ and $S_{ij} \in (0, 1, -1)$. $S_{ij}$ is 1 if document $d_i$ is more relevant than document $d_j$, -1 is document $d_j$ is more relevant than document $d_i$ and 0 if they have the same label. Substituting $\overline{P_{ij}}$ in equation (3.8) and combining it with equation (3.7), gives the following expression of cross entropy function to be minimized in training:

$$C = \frac{1}{2}(1 - S_{ij})\sigma(s_i - s_j) + log(1 + e^{-\sigma(s_i - s_j)}) \tag{3.9}$$

This cross entropy function is then minimized using Stochastic Gradient Descent[12]. The original paper applies RankNet to a shallow network. In our work we have extended the RankNet implementation to a deep network using batch learning as discussed in [5].

In terms of data, since the training doesn't require a true score, we just use the pair-wise relevance to train the model at every step. For a query $q$, if we have a set of documents $D = (d_1, ..., d_n)$, assuming $d_1$ is the relevant document, $d_1$ is paired with all the remaining documents from $D$ and each of the pair (along with the vector representation) the input to the learning process. Similar to the previous model, each $(user, query, document)$ triple is represented by a vector of size 600 (by concatenating 3 vectors of size 200).

As explained in [5], we now present how the batch learning works in the RankNet algorithm. For a given pair of document $d_i$ and $d_j$, the derivative of the cost function $C$ with respect to a weight $w_k$ is:

$$\frac{\partial C}{\partial w_k} = \frac{\partial C}{\partial s_i}\frac{\partial s_i}{\partial w_k} + \frac{\partial C}{\partial s_j}\frac{\partial s_j}{\partial w_k} = \sigma(\frac{1}{2}(1-S_{ij}) - \frac{1}{1+e^{\sigma(s_i-s_j)}})(\frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k}) = \lambda_{ij}(\frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k}) \tag{3.10}$$

Where we have defined $\lambda_{ij}$ as:

$$\lambda_{ij} = \sigma(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1+e^{\sigma(s_i-s_j)}}) \tag{3.11}$$

Also, the formula for the weight updates according to the Stochastic Gradient Descent[12] is:

$$w_k \rightarrow w_k - \eta\frac{\partial C}{\partial w_k} \tag{3.12}$$

For a set P which contains pairs $(i, j)$ such that $d_i$ and $d_j$ have different relevance orders, let's assume without loss of generality that for a pair $(i, j)$, $d_i$ is ranked higher than $d_j$. Then using the equation 3.12, by importing the values form 3.10 and 3.11, the final update value can be written as:

$$\delta w_k = -\eta \sum_{(i,j)\in P} (\lambda_{ij}(\frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k})) = -\eta \sum_i \lambda_i \frac{\partial s_i}{\partial w_k} \tag{3.13}$$

Where $\lambda_i$ is calculated as the increment by $\lambda_{ij}$ in case where the pair is $(i, j)$ and decrement by $\lambda_{ki}$ when the pair is $(k, i)$. We can do this because the partial derivatives of the scores $s_i$ with respect to the weight $w_k$ will be independent of the current value of $w_k$. This gives us:

$$\lambda_i = \sum_{j:(i,j)\in P} \lambda_{ij} - \sum_{j:(j,i)\in P} \lambda_{ji} \tag{3.14}$$

RankNet optimization can become very time consuming as the number of nodes and neurons grow in the network. Initial implementation of the algorithm uses the updates for each pair of documents which is very expensive in terms of computation time. Equation 3.13 and 3.14 suggests that we can combine the losses of multiple pairs together to update the weights. This is the batch learning that we have implemented in our work with a batch size of 128. The improvement in the computation time is very significant. Other batch sizes do not change the performance of the algorithm much hence all the results in the next section are based on the batch size of 128. Experimentation results are presented in the next chapter.

## 3.4 Margin RankNet

The third model we propose is an extension of [26], here we apply a pair-wise learner which tries to learn an implausibility function $f$. The implausibility score calculated for a triple of $(user, query, document)$ represents the unlikeliness of the document being relevant as in [26]. Hence, given a query $q$, if for a document $d_i$ the implausibility score is lower than for a document $d_j$, then $d_i$ is more relevant than $d_j$. We have used a loss function similar to the function used in [26]. We separate the steps of learning of user embedding and the learning of scoring function $f$. We use the user profiles learnt in [26] and include these in the input of the deep network model and again optimize a similar margin loss function. Training with the margin based loss function essentially maximizes the distance between the relevant and irrelevant documents. The function we used is:

$$L = \sum_{q \in Q} \sum_{(d_{rel}, d_{irrel}) \in P} max(0, \gamma + f(u, q, d_{rel}) - f(u, q, d_{irrel})) \tag{3.15}$$

Q is the set of all queries. For a query $q$, P is a set of pairs of documents such that one document is relevant and other is irrelevant. $\gamma$ is the margin hyper parameter. In terms of data, the training data is utilized for the pair-wise learning such that one document is relevant and other is irrelevant, similar to the RankNet model. The function learnt then is used to calculate new ranking on the test data which in turn is utilized to calculate the Precision and MRR. Now we analyze how the updates work in the optimization. For a weight $w_k$, the update equation in SGD[12] is:

$$w_k \to w_k - \eta \frac{\partial L}{\partial w_k} \tag{3.16}$$

rewriting L as

$$L = \sum_{q \in Q} \sum_{(d_{rel}, d_{irrel}) \in P} L^{'} \tag{3.17}$$

where

$$L^{'} = max(0, \gamma + f(u, q, d_{rel}) - f(u, q, d_{irrel})) \tag{3.18}$$

Taking the derivative,

$$\frac{\partial L^{'}}{\partial w_k} = \frac{\partial L^{'}}{\partial f(u, q, d_{rel})} \frac{\partial f(u, q, d_{rel})}{\partial w_k} + \frac{\partial L^{'}}{\partial f(u, q, d_{irrel})} \frac{\partial f(u, q, d_{irrel})}{\partial w_k} \tag{3.19}$$

Also,

$$\frac{\partial L^{'}}{\partial f(u, q, d_{rel})} = \begin{cases} 1, & \text{if } f(u, q, d_{irrel}) - f(u, q, d_{rel}) < \gamma. \\ 0, & \text{otherwise.} \end{cases} \tag{3.20}$$

$$\frac{\partial L^{'}}{\partial f(u, q, d_{irrel})} = \begin{cases} -1, & \text{if } f(u, q, d_{irrel}) - f(u, q, d_{rel}) < \gamma. \\ 0, & \text{otherwise.} \end{cases} \tag{3.21}$$

Combining equations 3.19, 3.20 and 3.21

$$\frac{\partial L'}{\partial f(u,q,d_{irrel})} = \begin{cases} \frac{\partial f(u,q,d_{rel})}{\partial w_k} - \frac{\partial f(u,q,d_{irrel})}{\partial w_k}, & \text{if } f(u,q,d_{irrel}) - f(u,q,d_{rel}) < \gamma. \\ 0, & \text{otherwise.} \end{cases} \tag{3.22}$$

$L^{'}$ is the cost for one pair of inputs. From equation 3.22, it is clear that the loss is only calculated when the difference between the implausibility value of two documents is smaller than the margin $\gamma$. Whenever this is the case, the loss is minimized until the difference is greater than the margin. In terms of the data, each triple $u_i, q_i, d_i$ is represented as a vector of size 600 (concatenating three vectors of size 200), where these embedding vectors are learnt as explained in the previous chapter and in [26]. The output $f$ is a function of these inputs learnt through pairwise training. The optimization is done using the Stochastic gradient descant[12]. We train the model for 100 epochs and multiple experiments regarding the network architecture are detailed in the next chapter.

# Chapter 4

# Experiments and Results

Following the methodology explained in the last chapter, here we present the experimentation results for the three models. First we present the dataset used in our work. Then for each model, we start with the experimentation on the network architecture and then present the effect of personalization. Finally, we compare the best possible architectures of each of the model with the baselines.

## 4.1 Dataset

As mentioned before, our work is in the extension of [26]. We are using the same data used in the mentioned paper. Also due to the restricted access to the text data, we have used the embeddings for user, query and documents directly (as provided by the authors of the paper). The original dataset consists of query logs of 106 anonymous users in 15 days from 01 July 2012 to 15 July 2012. The log entity contains a user identifier, a query log and top 10 documents. Due to the certain privacy policies, the authors of the original paper could only provide us with the processed log entry data. All the data is tokenized with user, query and document ids.

In terms of processing as explained in [26], it is argued that short term profiles perform better than the long term profiles. Short term profiles are constructed by using the users search pattern within a session and these profiles are then used to personalized search in the session. To identify a session, 30 minutes of inactivity is assumed to be the session end. For each session, last log entries are assigned to the test and validation set and the rest is used for training. By this procedure, 106 original users get converted to 1584 short term user profiles.

To identify the actual relevance of a document, SAT clicks [11] are utilized as explained in [26]. For example, given a query 10 documents are returned and the user clicks on a document, it is assumed to be a relevant document and it gets re-ranked to be higher in the document list. With this method, each set of documents returned by the search engine are re-ranked with pushing the clicked document towards the top. The data provided to us contains the final re-ranked list for each of the user profile with the search engine ranks also indicated. To summarize, for each user profile and query, we have a set of document which are ranked in order of actual

| Stat | Value |
|---|---|
| Number of Days | 15 |
| Number of Actual Users | 106 |
| Number of User Profiles | 1584 |
| Number of Distinct Queries | 6632 |
| Number of Distinct Documents | 33591 |

Table 4.1: Description of the Search Engine Data[26]

relevance, each of the documents have their search engine rank associated with them. Also, if a user clicks more than one document, the whole document set is presented again with the clicked document on the top (as relevant) and other documents below the top (as irrelevant). The re-ranking is done in a binary manner with only one relevant document being considered at a time. Considering $(query, user, document)$ as a triple, there are 5685 correct (relevant) triples in the training data. Test and validation sets contain 1210 and 1184 correct triples, respectively. Table 4.1 gives an overview of the search engine data.

As mentioned before, because of the limited availability of the data, we are using the embeddings directly from the authors of the paper. We were provided with the vector of size 200 for all the documents, queries and user profiles. We directly utilize these representations in our work on personalized learning to rank problem.

## 4.2 Point-wise Scoring Model

### 4.2.1 Experiments

In this section, we discuss the different variations we tried to figure out the best possible configuration of the scoring model. First we experiment on the number of hidden layer and the number of nodes keeping the loss function to be MSE. We apply some regularization to study the effects on the precision and MRR. Then we change the loss function and study the performance.

On the architecture side, we test with 1, 2 and 3 layer variants with the configurations mentioned in table 4.2 with a config number assigned to each case. The evaluation is in terms of the $P@1$ and $MRR$ metrics defined before. The model is trained with 10 epochs and with a batch size of 128. Other epochs and batch sizes don't give any improvement in the performance.

Figure 4.1 depicts the performance of each of the configurations listed in the table 4.2 on the validation set. Config 4, which is a 3-hidden layer 128-64-32 neuron architecture performs better in terms of both MRR and Precision. Other configurations were also run but with little or no change in the metrics. Next we apply a dropout layer in all the tested configurations. We test with a dropout probability of 0.25. Figure 4.2 depicts the performance of the configurations with the dropout. For the config 4, introducing a dropout layer actually makes the performance worse. Dropout in other

| #Config | #Input Neurons | #Hidden Layers | #Neurons | #Output Neurons |
|---|---|---|---|---|
| 1 | 600 | 1 | 32 | 1 |
| 2 | 600 | 1 | 64 | 1 |
| 3 | 600 | 2 | 64-32 | 1 |
| 4 | 600 | 3 | 128-64-32 | 1 |

TABLE 4.2: Configuration of Different Architectures for the Scoring Model

configuration doesn't effect much. Hence, we do the rest of the parameter tuning on the selected configuration 4.

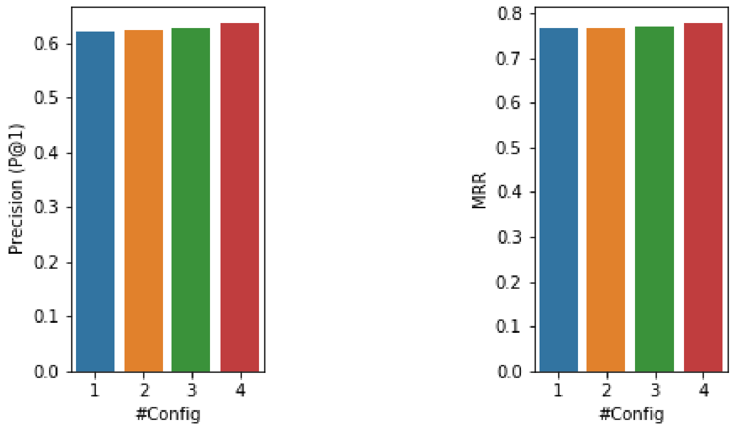

FIGURE 4.1: Comparison of performances of the number of Layers and Neurons on Precision(P@1) and MRR for Scoring Model

Next we analyze different loss function suitable for a regression problem on the selected configuration. Until now the calculations are based on the Mean Squared Error. We study the effect of changing the loss function to Mean Absolute Error, Mean Squared log Error and logcosh. Figure 4.3 depicts the performance of each of these loss functions in terms of Precision and MRR. The performance of all the four loss functions is more of less the same with MSE being slightly better than the rest. This shows that any of these functions can be used as loss in our model. We move on with the MSE.

With a dropout probability of 0.25, we did not see any improvement in the performance. Hence, we try with different dropout probabilities on the selected configuration of the 3-layered network. Figure 4.4 shows the precision and MRR as a function of the dropout probability $p$ varying between 0.1 and 0.5. We again see than none of the dropout probabilities improve the performance over the original model. This means that the model is not over-fitting, which could be because the 3-layered model is not deep enough.

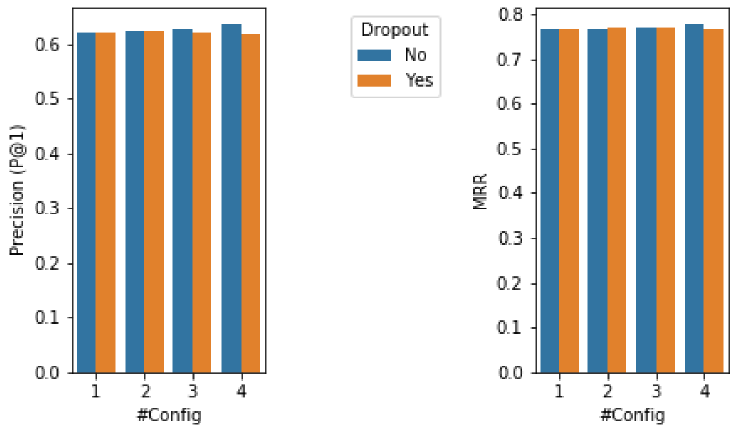

Figure 4.2: Comparison of performances of the number of Layers and Neurons on Precision(P@1) and MRR with dropout for Scoring Model

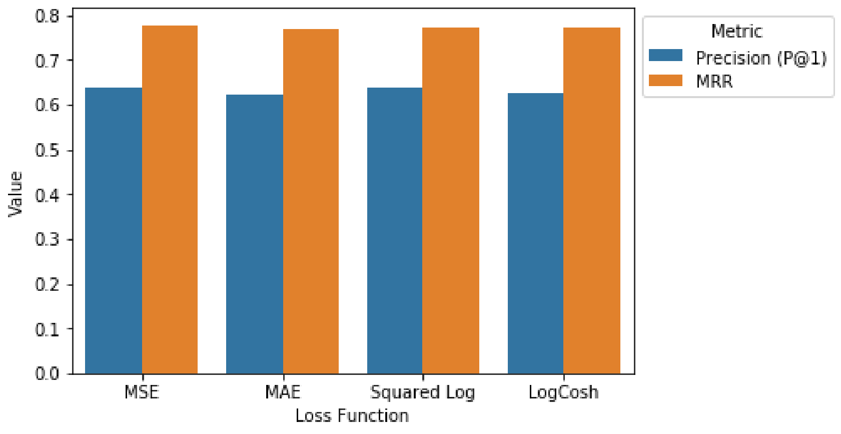

Figure 4.3: Comparison of performances of different loss functions on Precision(P@1) and MRR for Scoring Model

### 4.2.2 Effect of Personalization

In this subsection, we study the effect of using the user embeddings on the model. We compare the performance of the models with and without user embeddings as part of the input. Owing to the discussion in the previous subsection, we have established that the ideal configuration of the neural architecture in the point-wise scoring model is of 3 hidden layers with 128 neurons in the first layer, 64 neurons in the second and 32 in the third layer and the selected loss function is the MSE. Until this point, we have included the user embedding in the input of the training set. Now, to check if the effect of user embedding is significant of not, we retrain a new model with the same configurations but a different input structure. The input in this new model

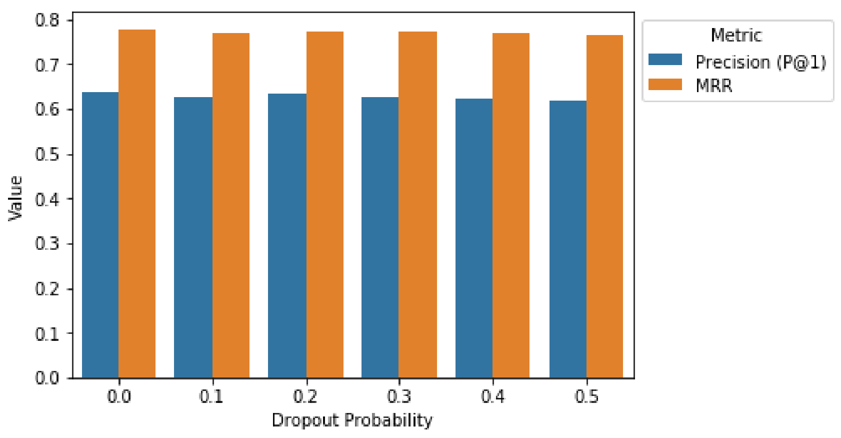

Figure 4.4: Effect of different dropout probabilities on Precision(P@1) and MRR for Scoring Model

contains only the query and document embeddings. The parameter tuning until the last section was on the validation data but here we compare the performance on the test data. Figure 4.5 depicts the comparison between the two models in terms of Precision and MRR. We have performed welch's t-test[A.1] to test the significance of the difference between the two models. For precision, we get a p-value of 0.002 and for MRR the p-value is 0.0034. For both of the metrics, p-value $< 0.05$, hence we can say that the difference in the performance is statistically significant. To conclude, the point-wise scoring model works better when we use the user embedding vector as part of the input. The comparison of this model with the baseline is detailed later in this chapter.

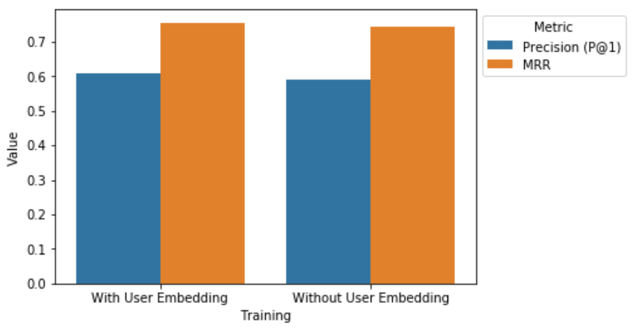

Figure 4.5: Effect of Personalization on the Ranking for Scoring Model

| #Config | #Hidden Layers | #Neurons | Precision (P@1) | MRR |
|---|---|---|---|---|
| 1 | 1 | 128 | 0.5615 | 0.7204 |
| 2 | 1 | 256 | 0.5698 | 0.7326 |
| 3 | 1 | 512 | 0.5966 | 0.7483 |
| 4 | 2 | 128-64 | 0.6097 | 0.7595 |
| 5 | 2 | 256-128 | 0.6167 | 0.7638 |
| 6 | 2 | 512-128 | 0.6167 | 0.7647 |
| 7 | 3 | 128-64-32 | 0.6034 | 0.7557 |
| 8 | 3 | 256-128-64 | 0.6138 | 0.7585 |

Table 4.3: Configuration of Different Architectures for Deep RankNet

## 4.3 Deep RankNet

Following subsections detail the experimentation for our RankNet implementation. First we present the experiments for the configuration selection, then we study the effect of personalization on the model performance.

### 4.3.1 Experiments

We tested the performance of different models by varying the number of layers and neurons. We vary the number of layer from 1 to 3 with different number of neurons. We try to decrease the number of neurons in the subsequent layers. Table 4.3 gives the precision and MRR value of eight different architectures that are significant in terms of number of hidden layers, number of neurons. A few other configurations have been tried as well, the experiments show that there was almost no impact on the performance compared to the ones already presented. Figure 4.6 show the precision and MRR for each of the configurations. Networks with just 1 layer are clearly outperformed by the 2-hidden layer networks. The 2-hidden layer and 3-hidden layer networks have very similar performance in terms of precision and MRR. The 2-hidden layer network with 512 neurons in the first and 128 neurons in the second layer gives the best result with P@1 of 0.6167 and MRR of 0.7647 on the validation data. Moving forward, we pick the 2-layer 512-128 neurons case as the best architecture. The model here is trained on 50 epochs, lower number of epochs reduce the performance and higher number of epochs don't give any significant improvement.

In our chosen architecture of 2 hidden layers, we try and apply regularization by adding a dropout layer after the first layer. We vary the dropout probability in the range of $(0, 0.6]$ to study the effect on the P@1 and MRR. Figure 4.7 illustrates the precision and MRR both increase for the dropout probability of 0.2 and 0.3. Dropout probability of 0.3 gives the best performance. Hence, we utilize this configuration for testing the performance on the test data in the next subsection.

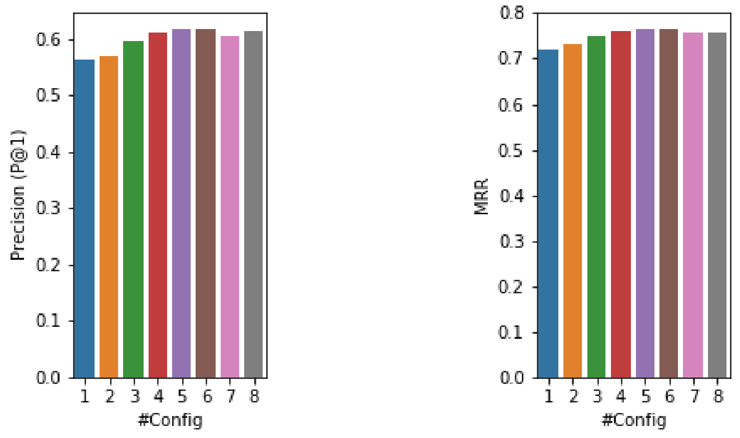

FIGURE 4.6: Comparison of performances of different configurations on Precision(P@1) and MRR for Deep RankNet

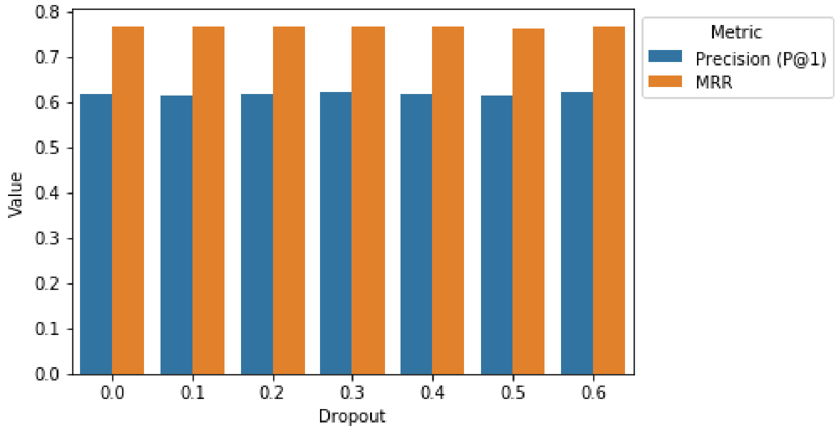

FIGURE 4.7: Effect of different dropout probabilities on Precision(P@1) and MRR for Deep RankNet

### 4.3.2 Effect of Personalization

The network architecture we have selected is a model with 2 hidden layers (512-128) and a dropout of 0.3. In this section, instead of an input size of 600, we train the same model by removing the user embedding vector from the input and using a size 400 vector. Figure 4.8 shows the average value of P@1 and MRR over 5 independent runs with both input models (with and without embedding) on the test dataset.

The model with user embeddings seems to perform slightly better. To test the statistical significance in the difference, we applied welch's t-test[A.1]. The p-value for P@1 is 0.008 and for MRR is 0.04, both of which smaller than 0.05, hence we can say the difference between two sets is significant for both the metrics. To Conclude,

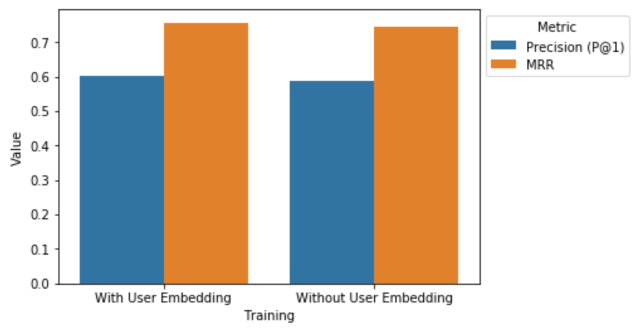

Figure 4.8: Effect of Personalization on the Ranking for Deep RankNet

the ideal architecture for the our RankNet implementation turns out to be 2-hidden layers network with a dropout of 0.3. The ranking is significantly better if we use the user embedding as a part of the input.

## 4.4 Margin RankNet

Following subsections detail the experimentation on the proposed Margin RankNet model. First we present the experiments related to the network configuration, then we study the effect of personalization on the model performance.

### 4.4.1 Experiments

In this section we utilize the validation dataset to find the best possible configurations for the model. The parameters to tune are the number of layers, neurons and value of $\gamma$, the margin hyper parameter. We chose the hyper-parameter $\gamma \in 1, 3, 5$ as done in [26]. Table 4.4 details the different architectures that we tried along with the precision and MRR for each case. Results show that 1 hidden layer networks perform significantly worse than 2 and 3 hidden layered models, with a precision of around 0.5 and MRR of 0.65. Hence, we are only presenting results for 2 and 3 hidden layered architectures. For each of the values of $\gamma$, we ran the model for 100 epochs with 4 different architectures. Figure 4.9 depicts the precision and MRR trend for each of the experiments. Precision for the three layered architecture with 256-128-64 neuron structure performs really well for all the values of $\gamma$. for MRR, the 3 layered case for each $\gamma$ is very similar. Hence, we choose the 3-layered architecture of 256-128-64 neurons with a $\gamma$ of 5 to be the best architecture.

In our chosen architecture of 3-hidden layers, we now apply some regularization by introducing dropout layers after the first and second hidden layer. As before, we plot the precision and MRR as a function of dropout probability $p$ in figure 4.10 with $p \in (0, 0.6]$. The precision increases for dropout probabilities 0.1, 0.2 and 0.3. With a probability of 0.2, the precision is 0.7282 which is significantly better than the non-regularized architecture. MRR drops for all the choices of $p$ but the improvement

| $\gamma$ | **#Layers** | **#Neurons** | **P@1** | **MRR** |
|---|---|---|---|---|
| 1 | 2 | 256-128 | 0.6582 | 0.7371 |
| 1 | 2 | 512-128 | 0.6751 | 0.7443 |
| 1 | 3 | 128-64-32 | 0.684 | 0.743 |
| 1 | 3 | 256-128-64 | 0.6982 | 0.7471 |
| 3 | 2 | 256-128 | 0.6553 | 0.7386 |
| 3 | 2 | 512-128 | 0.6718 | 0.746 |
| 3 | 3 | 128-64-32 | 0.6698 | 0.7386 |
| 3 | 3 | 256-128-64 | 0.6923 | 0.7479 |
| 5 | 2 | 256-128 | 0.6632 | 0.7413 |
| 5 | 3 | 512-128 | 0.6801 | 0.7452 |
| 5 | 3 | 128-64-32 | 0.6571 | 0.7373 |
| 5 | 3 | 256-128-64 | 0.7012 | 0.7437 |

TABLE 4.4: Configuration of Different Architectures for Margin RankNet

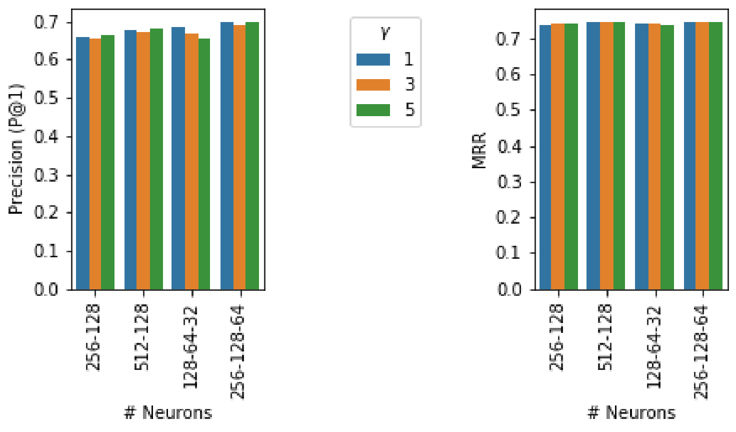

FIGURE 4.9: Comparison of performances of different configurations on Precision(P@1) and MRR for Margin RankNet

in precision is substantial for $p = 0.2$. Hence, this we chose the 3-layered network with a dropout probability of 0.2 to be the best possible architecture.

### 4.4.2 Effect of Personalization

The network architecture which performs the best after the parameter tuning and regularization is 3-hidden layered model with 256 neurons in the first hidden layer 128 in the seconds and 64 in the third with a dropout probability of 0.2 in the first two hidden layers. In this section, we compare the performance of the models with and without user embeddings in the input vector on the test set. We train a new model with the input size of 400 (query and document). Figure 4.11 shows the performance

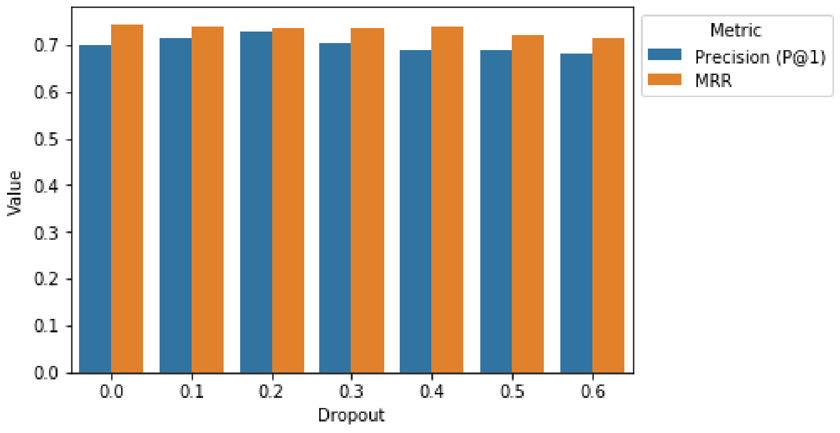

Figure 4.10: Effect of different dropout probabilities on Precision(P@1) and MRR for Margin RankNet

of the two models in terms of P@1 and MRR averaged over 5 independent runs.

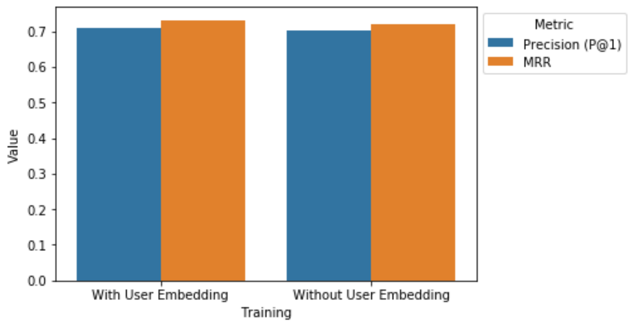

Figure 4.11: Effect of Personalization on the Ranking RankNet

The model with user embedding is performing slightly better. We have a applied welch's t-test[A.1] to test the statistical significance of the difference. The p-value for P@1 is 0.006 and for MRR is 0.0003. Both p-values are less than 0.05, hence we can say that the model with the user embedding in the input vector performs significantly better than the model without user embedding. To conclude, the best architecture of the Margin RankNet model is the 3-hidden layered model with a dropout probability of 0.2. This model performs significantly better than the variant with no user embedding in the input vector.

## 4.5 Comparison with Baselines

In this part, we summarize the most substantial results obtained in the last three sections and compare them with the baselines. Table 4.5 presents the network archi-

| Property | Scoring Model | Deep RankNet | Margin RankNet |
|---|---|---|---|
| # Layers | 3 hidden Layers | 2 Hidden Layers | 3 Hidden Layers |
| # Neurons | 128-64-32 | 512-128 | 256-128-64 |
| Regularization | No Regularization | Dropout = 0.3 | Dropout = 0.2 |
| Loss | MSE | Cross Entropy | Margin Loss |

TABLE 4.5: Architecture of Models used for comparison. ReLU activation is used in the hidden layers in all the three models

| Model | P@1 | MRR |
|---|---|---|
| SE | 0.385 | 0.559 |
| Baseline1 | 0.501 | 0.656 |
| Baseline2 | 0.481 | 0.645 |
| Scoring Model | $0.6071_{21\%,26\%}$ | $0.7551_{15\%,17\%}$ |
| Scoring Model(No User) | $0.59_{18\%,23\%}$ | $0.743_{13\%,15\%}$ |
| Deep RankNet | $0.6024_{20\%,25\%}$ | $0.7571_{15\%,17\%}$ |
| Deep RankNet(No User) | $0.5875_{17\%,22\%}$ | $0.7457_{14\%,16\%}$ |
| Margin RankNet | $0.7108_{42\%,48\%}$ | $0.7341_{11\%,13\%}$ |
| Margin RankNet(No User) | $0.7014_{40\%,46\%}$ | $0.7194_{10\%,12\%}$ |

TABLE 4.6: Performances of the models on the test set. Comparison with the three baselines. The subscripts represents the improvement on from Baseline1 and Baseline2. "No User" in the model name represent that user embeddings weren't used in training

tectures used for the three models selected through experimentation. We compare the results of the three models with the baselines - Search Engine and Methods suggested in [26] in Table 4.6 along with the variants without user embeddings. Baseline1 is the original model in [26] with randomized weight matrices, Baseline2 is the simplified model where the matrices are fixed as identity. SE represents the ranking from the search engine.

It is clear from the results that using deep neural architectures for training ranking models drastically improve the performance. Also, combining embedding features in itself gives significant improvement in the ranking quality. All the three models that we presented outperform the baseline models significantly. Even without the use of user embeddings, we see the precision improving by more than 17% and and MRR by more than 13%. Using the text embeddings as features, our models are able to detect the semantic relations between the query and documents. Using the user profiles significantly improves the performance of all the three ranking models. The pointwise scoring model and the deep RankNet model don't have a very significant difference in terms of performance. But, the Margin RankNet model gives a significant improvement on both Scoring model by 17% and Deep RankNet by 18% in terms of precision. This shows that our Margin RankNet model is significantly more precise than these two models. Using a margin based loss function leads to better results. The personalization in all the three models is improves the ranking significantly with the deep learning framework leading to substantially better results compared to baselines.

# Chapter 5

# Conclusion

We started this thesis with the aim of developing a solution for the personalized information retrieval problem by creating learning to rank solutions in a deep learning framework. We tackled the problem by using the text embedding vectors as the features. Only a few algorithm have utilized this approach before. The algorithms which do work with the embedding data don't focus on the personalization of results by using the user profiles. Also, there has been limited work on the deep learning framework for learning to rank problem. The algorithm in [26] learns the ranking of documents and user profiles together by utilizing a margin based loss function. Our work is in extension of the mentioned paper. We directly utilized the query, document and user embeddings into three different deep learning models which outperformed the baselines.

We applied a pointwise scoring model, implemented the existing RankNet model in a deep learning framework and proposed a new model, Margin RankNet model. The results of our experiments show that all these three models significantly outperform the baseline models. First model was a pointwise deep learning model which uses MSE as a loss function. With some experiments, we established that a three layered model performs the best. The second model is just a deep network implementation of the existing RankNet algorithm. We utilized batch learning here to speed up the learning process substantially. Finally, the third model utilizes a pairwise training approach by using the margin based loss function used in the profile creation step in [26]. The two level of optimization (first in profile creation and then in ranking), provides us with extremely good results. All of these strategies perform significantly better than the original method of [26]. Using the embeddings data in the learning to rank problem we establish that the models are able to learn semantic abstractions between the query and documents.

Our work in this thesis was based on the data created by the authors of [26] as due to the privacy policy we couldn't get the original data. Also, to the best of our knowledge, the available data sets on the web do not provide enough information with the true ranking orders. The open source data sets, which contain raw text data for query and users, either do not provide the full list of returned document [1] (which

[1] http://www.cim.mcgill.ca/~dudek/206/Logs/AOL-user-ct-collection/

is required train the irrelevancy) or they are missing the user information[2] (which is required for personalization). Another way was to manually label the returned documents from the existing data sets into correct order of relevance, which was not feasible given the resources. Hence, we decided to go with the already created profiles. Though as a potential future work, given the data, it would be an interesting study to create the user profiles and ranking function simultaneously using deep learning framework. User embeddings can also be independently learnt using the methods described in [2] where they use a Generalized Canonical Correlation Analysis or in [1] where they use deep learning to learn user embedding based on user related texts (twitter posts). These user embedding creation models can be utilized in the ranking model independently. Also, in terms of the ranking model, it would be interesting to study how other widely used state-of-the-art algorithms perform with the embedding data.

[2] http://mlr.cs.umass.edu/ml/machine-learning-databases/ohsumed/

# Appendices

# Appendix A

# Welch's t-test

## A.1 Welch's t-test

Welch's t-test[1] or unequal variance t-test is a statistical test to test the hypothesis that two set of populations have equal means. This is an important analysis in terms of comparing the outputs of different models which are run independently. In the next chapter, this test is used to analyze the significance of the user embedding vector on the models. The test defines the t-statistics as follows:

$$t = \frac{\mu_1 - \mu_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}} \tag{A.1}$$

Where $\mu_1$ and $\mu_2$ are the sample means of the two samples, $s_1$ and $s_2$ are sample variances and $N_1$ and $N_2$ are the sample sizes. The degree of freedom $\upsilon$ is estimated as:

$$\upsilon = \frac{(\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2})^2}{\frac{s_1^4}{N_1\upsilon_1} + \frac{s_2^4}{N_2\upsilon_2}} \tag{A.2}$$

Where $\upsilon_1 = N_1 - 1$ and $\upsilon_2 = N_2 - 1$ are the degrees of freedom associated with sample 1 and 2 respectively. With these estimates of the t statistics and the degrees of freedom, following null hypothesis $H_0$ can be tested with the t-distribution. If the p-value is less than 0.05, the we reject the null hypothesis $H_0$, otherwise we accept it.

1. $H_0 : \mu = \mu_1 - \mu_2 = 0$
2. $H_1 : \mu = \mu_1 - \mu_2 \neq 0$

[1] https://en.wikipedia.org/wiki/Welch%27s_t-test

# Bibliography

[1] S. Amir, G. Coppersmith, P. Carvalho, M. J. Silva, and B. C. Wallace. Quantifying mental health from social media with neural user embeddings. *arXiv preprint arXiv:1705.00335*, 2017.

[2] A. Benton, R. Arora, and M. Dredze. Learning multiview embeddings of twitter users. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 14–19, 2016.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[4] 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*, pages 89–96. ACM, 2005.

[5] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

[6] C. J. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *Advances in neural information processing systems*, pages 193–200, 2007.

[7] 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*, pages 129–136. ACM, 2007.

[8] A. Cherif, S. Jouili, and G. Huybrechts. Deep neural networks for learning to rank. *BENELEARN*, 2016.

[9] K. Crammer and Y. Singer. Pranking with ranking. In *Advances in neural information processing systems*, pages 641–647, 2002.

[10] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[11] S. Fox, K. Karnawat, M. Mydland, S. Dumais, and T. White. Evaluating implicit measures to improve web search. *ACM Transactions on Information Systems (TOIS)*, 23(2):147–168, 2005.

[12] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[13] T. Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 289–296. Morgan Kaufmann Publishers Inc., 1999.

[14] T.-Y. Liu. *Learning to rank for information retrieval.* Springer Science & Business Media, 2011.

[15] L. Pang, Y. Lan, J. Guo, J. Xu, J. Xu, and X. Cheng. Deeprank: A new deep architecture for relevance ranking in information retrieval. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 257–266. ACM, 2017.

[16] V. Patel. Kalman-based stochastic gradient method with stop condition and insensitivity to conditioning. *SIAM Journal on Optimization*, 26(4):2620–2648, 2016.

[17] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[18] L. Rigutini, T. Papini, M. Maggini, and F. Scarselli. Sortnet: Learning to rank by a neural preference function. *IEEE transactions on neural networks*, 22(9):1368–1380, 2011.

[19] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[20] A. Severyn and A. Moschitti. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 373–382. ACM, 2015.

[21] B. Song. Deep neural network for learning to rank query-text pairs. *arXiv preprint arXiv:1802.08988*, 2018.

[22] Y. Song, H. Wang, and X. He. Adapting deep ranknet for personalized search. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 83–92. ACM, 2014.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[24] J. Suykens. Artificial neural networks. *ESAT-STADIUS*, 2013.

[25] D. van Ravenzwaaij, P. Cassey, and S. D. Brown. A simple introduction to markov chain monte–carlo sampling. *Psychonomic bulletin & review*, 25(1):143–154, 2018.

[26] T. Vu, D. Q. Nguyen, M. Johnson, D. Song, and A. Willis. Search personalization with embeddings. In *European Conference on Information Retrieval*, pages 598–604. Springer, 2017.

[27] B. Wang and D. Klabjan. An attention-based deep net for learning to rank. *arXiv preprint arXiv:1702.06106*, 2017.

# Master's thesis filing card

*Student*: Kshitij Goyal

*Title*: Personalized Search with Deep Learning

*UDC*: 681.3*I20

*Abstract*:

With an ever-increasing consumption of digital information, learning to rank has become an important topic in computer science. In this thesis, we propose and develop a deep learning based learning to rank framework to deal with personalized information retrieval. There has been extensive work done on the topic of learning to rank. Because of an increasing interest in deep learning over the past few year, few works in the learning to rank have been proposed in deep learning framework. Deep learning has made significant improvement in the learning quality in some recent works. Deep networks are useful in discovering abstractions and identifying the relations between the documents and the user query. Despite the fact that user profiles are very important for personalized information retrieval, only a few of the existing works deal with the application of user profiles in the deep learning ranking models. In this thesis we propose the user profile-based learning to rank methods - Pointwise Scoring Model, Deep RankNet and Margin RankNet, and evaluate the frameworks in a personalized information retrieval setting. Our work shows that personalization of ranking using user profiles significantly improves the ranking of documents when we use the deep learning framework, with the Margin RankNet model performing better than both pointwise scoring model and Deep RankNet model in terms of precision.

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Big Data Analytics

*Thesis supervisor*: Prof. Dr. Marie-Francine Moens

*Assessors*: Katrien Laenen
Dr. Robin De Croon

*Mentor*: Dr. Quynh Ngoc Thi Do